sun®
microsystems

# SunOS™ Reference Manual

Part Number: 800-1751-10
Revision A, of 9 May 1988

## Credits and Trademarks

Sun Workstation® is a registered trademark of Sun Microsystems, Inc.

SunStation®, Sun Microsystems®, SunCore®, SunWindows®, DVMA®, and the combination of Sun with a numeric suffix are trademarks of Sun Microsystems, Inc.

UNIX, UNIX/32V, UNIX System III, and UNIX System V are trademarks of AT&T Bell Laboratories.

Intel® and Multibus® are registered trademarks of Intel Corporation.

DEC®, PDP®, VT®, and VAX® are registered trademarks of Digital Equipment Corporation.

## NAME

intro – introduction to commands

## DESCRIPTION

This section describes publicly accessible commands in alphabetic order. Commands of general utility, many with enhancements from 4.3 BSD. Wherever possible, we have incorporated System V versions of commands and utilities into our standard UNIX release. Where a command has both a System V and a BSD version and it has been possible to merge them, we have done so. In some cases, where the System V version is compatible with BSD and offers significant added value, SunOS incorporates that version as its standard.

Pages of special interest have been categorized as follows:

1C Commands for communicating with other systems.

1G Commands used primarily for graphics and computer-aided design.

1V Commands that only have System V versions, or that have separate BSD and System V versions.

These commands either depend upon System V functionality, or have incompatibilities with the corresponding BSD version. They are included in the System V Software installation option. Once installed, they can be found in the directory **/usr/5bin**.

## SEE ALSO

- Section 8 in this manual for system administration procedures, system maintenance and operation commands, local daemons, and network-services servers.
- Section 7 in this manual for descriptions of publicly available files and macro packages for document preparation.
- Section 6 in this manual for computer games.
- *Getting Started with SunOS: Beginner's Guide*
- *Setting Up Your SunOS Environment: Beginner's Guide*
- *SunView 1 Beginner's Guide*
- *Using the Network: Beginner's Guide*
- *Doing More with SunOS: Beginner's Guide*
- *Programming Utilities and Libraries*

## DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of "normal" termination) one supplied by the program, see **wait** and **exit**(2). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code," "exit status" or "return code," and is described only where special conventions are involved.

LIST OF COMMANDS

| Name | Appears on Page | Description |
|---|---|---|
| % | csh(1) | C shell built-in commands |
| @ | csh(1) | C shell built-in commands |
| Mail | mail(1) | read or send mail messages |
| adb | adb(1) | general-purpose debugger |
| addbib | addbib(1) | create or extend a bibliographic database |
| adjacentscreens | adjacentscreens(1) | connect multiple screens to SunView window driver |
| admin | admin(1) | create and administer SCCS files |
| aedplot | plot(1G) | graphics filters for various plotters |
| alias | csh(1) | C shell built-in commands |
| align_equals | textedit_filters(1) | filters provided with textedit(1) |
| ar | ar(1V) | create library archives, and add or extract files |
| arch | arch(1) | display the architecture of the current host |
| as | as(1) | Sun-1, Sun-2 and Sun-3, Sun-4 and Sun386i assemblers |
| at | at(1) | execute a command or script at a specified time |
| atq | atq(1) | display the queue of jobs to be run at specified times |
| atrm | atrm(1) | remove jobs spooled by at or batch |
| awk | awk(1) | pattern scanning and processing language |
| banner | banner(1V) | display a string in large letters |
| bar | bar(1) | create tape archives, and add or extract files |
| basename | basename(1) | display portions of pathnames and filenames |
| batch | at(1) | execute a command or script at a specified time |
| bc | bc(1) | arbitrary-precision arithmetic language |
| bg | csh(1) | C shell built-in commands |
| bgplot | plot(1G) | graphics filters for various plotters |
| biff | biff(1) | give notice of incoming mail messages |
| binmail | binmail(1) | an early program for processing mail messages |
| break | csh(1) | C shell built-in commands |
| breaksw | csh(1) | C shell built-in commands |
| cal | cal(1) | display a calendar |
| calendar | calendar(1) | a simple reminder service |
| capitalize | textedit_filters(1) | filters provided with textedit(1) |
| case | csh(1) | C shell built-in commands |
| cat | cat(1V) | concatenate and display |
| cb | cb(1) | a simple C program beautifier |
| cc | cc(1V) | C compiler |
| cd | cd(1) | change working directory |
| cdc | cdc(1) | change the delta commentary of an SCCS delta |
| cflow | cflow(1) | generate a flow graph for a C program |
| checkeq | eqn(1) | typeset mathematics |
| checknr | checknr(1) | check nroff and troff input files; report possible errors |
| chfn | passwd(1) | change password file information |
| chgrp | chgrp(1) | change the group ownership of a file |
| chkey | chkey(1) | change your encryption key |
| chmod | chmod(1V) | change the permissions mode of a file |
| chsh | passwd(1) | change password file information |
| clear | clear(1) | clear the terminal screen |
| clear_colormap | clear_colormap(1) | clear the colormap to make console text visible |
| clear_functions | clear_functions(1) | reset the selection service to clear stuck function keys |
| click | click(1) | enable or disable the keyboard's keystroke click |
| clock | clock(1) | display the time in an icon or window |

| | | |
|---|---|---|
| **cluster** | **cluster**(1) | find the Sun386i cluster containing a file |
| **cmdtool** | **cmdtool**(1) | run a shell (or program) using the SunView text facility |
| **cmp** | **cmp**(1) | perform a byte-by-byte comparison of two files |
| **col** | **col**(1V) | filter reverse paper motions from nroff for display |
| **colcrt** | **colcrt**(1) | filter nroff output for a terminal lacking overstrike |
| **coloredit** | **coloredit**(1) | alter color map segment |
| **colrm** | **colrm**(1) | remove characters from specified columns within each line |
| **comb** | **comb**(1) | combine SCCS deltas |
| **comm** | **comm**(1) | display lines in common between two sorted lists |
| **compress** | **compress**(1) | compress or expand files, display expanded contents |
| **continue** | **csh**(1) | C shell built-in commands |
| **cp** | **cp**(1) | copy files |
| **cpio** | **cpio**(1) | copy file archives in and out |
| **cpp** | **cpp**(1) | the C language preprocessor |
| **crontab** | **crontab**(1) | install, edit, remove or list a user's crontab file |
| **crtplot** | **plot**(1G) | graphics filters for various plotters |
| **crypt** | **crypt**(1) | encode or decode a file |
| **csh** | **csh**(1) | a shell with a C-like syntax |
| **csplit** | **csplit**(1) | split a file with respect to a given context |
| **ctags** | **ctags**(1) | create a tags file for use with vi |
| **ctrace** | **ctrace**(1) | generate a C program execution trace |
| **cu** | **tip**(1C) | terminal emulator, telephone connection to a remote system |
| **cut** | **cut**(1) | remove selected fields from each line of a file |
| **cxref** | **cxref**(1) | generate a C program cross-reference |
| **date** | **date**(1V) | display or set the date |
| **dbx** | **dbx**(1) | source-level debugger |
| **dbxtool** | **dbxtool**(1) | SunView interface for the dbx source-level debugger |
| **dc** | **dc**(1) | desk calculator |
| **dd** | **dd**(1) | convert and copy files with various data formats |
| **default** | **csh**(1) | C shell built-in commands |
| **defaults_from_input** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **defaults_from_input** | **input_from_defaults**(1) | update the mouse and keyboard from defaults |
| **defaults_to_indentpro** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **defaults_to_mailrc** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **defaultsedit** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **delta** | **delta**(1) | make a delta (change) to an SCCS file |
| **deroff** | **deroff**(1) | remove nroff, troff, tbl and eqn constructs |
| **des** | **des**(1) | encrypt or decrypt data using Data Encryption Standard |
| **df** | **df**(1) | report free disk space on file systems |
| **diff3** | **diff3**(1V) | display line-by-line differences between 3 files |
| **diff** | **diff**(1) | display line-by-line differences between pairs of text files |
| **diffmk** | **diffmk**(1) | mark differences between versions of a troff input file |
| **dircmp** | **dircmp**(1V) | compare directories |
| **dirname** | **basename**(1) | display portions of pathnames and filenames |
| **dirs** | **csh**(1) | C shell built-in commands |
| **dis** | **dis**(1) | object code disassembler for COFF |
| **domainname** | **domainname**(1) | set or display name of the current YP domain |
| **dos2unix** | **dos2unix**(1) | convert text file from DOS format to SunOS format |
| **dos** | **dos**(1) | SunView window for IBM PC/AT applications |
| **du** | **du**(1V) | display the number of disk blocks used per directory or file |
| **dumbplot** | **plot**(1G) | graphics filters for various plotters |
| **e** | **ex**(1) | line editor |
| **echo** | **echo**(1V) | echo arguments to the standard output |

| | | |
|---|---|---|
| **ed** | **ed**(1) | basic line editor |
| **edit** | **ex**(1) | line editor |
| **egrep** | **grep**(1V) | search a file for a string or regular expression |
| **else** | **csh**(1) | C shell built-in commands |
| **end** | **csh**(1) | C shell built-in commands |
| **endif** | **csh**(1) | C shell built-in commands |
| **endsw** | **csh**(1) | C shell built-in commands |
| **enroll** | **xsend**(1) | send or receive secret mail |
| **env** | **env**(1) | obtain or alter environment variables |
| **eqn** | **eqn**(1) | typeset mathematics |
| **error** | **error**(1) | categorize compiler error messages |
| **eval** | **csh**(1) | C shell built-in commands |
| **ex** | **ex**(1) | line editor |
| **exec** | **csh**(1) | C shell built-in commands |
| **exit** | **csh**(1) | C shell built-in commands |
| **expand** | **expand**(1) | expand TAB characters to SPACE characters |
| **expr** | **expr**(1V) | evaluate arguments as an expression |
| **false** | **true**(1) | provide truth values |
| **fg** | **csh**(1) | C shell built-in commands |
| **fgrep** | **grep**(1V) | search a file for a string or regular expression |
| **file** | **file**(1) | determine the type of a file by examining its contents |
| **find** | **find**(1) | find files by name, or by other characteristics |
| **finger** | **finger**(1) | display information about users |
| **fmt** | **fmt**(1) | simple text and mail-message formatters |
| **fmt_mail** | **fmt**(1) | simple text and mail-message formatters |
| **fold** | **fold**(1) | fold long lines for display |
| **fontedit** | **fontedit**(1) | a vfont screen-font editor |
| **foption** | **foption**(1) | determine available floating-point code generation options |
| **foreach** | **csh**(1) | C shell built-in commands |
| **from** | **from**(1) | display the sender and date of newly-arrived mail messages |
| **ftp** | **ftp**(1C) | file transfer program |
| **gcore** | **gcore**(1) | get core images of running processes |
| **get** | **get**(1) | get a version of an SCCS file |
| **get_selection** | **get_selection**(1) | copy contents of a selection to the standard output |
| **getopt** | **getopt**(1) | parse command options in shell scripts |
| **getoptcvt** | **getopts**(1) | parse command options in shell scripts |
| **getopts** | **getopts**(1) | parse command options in shell scripts |
| **gfxtool** | **gfxtool**(1) | run graphics programs in a SunView window |
| **gigiplot** | **plot**(1G) | graphics filters for various plotters |
| **glob** | **csh**(1) | C shell built-in commands |
| **goto** | **csh**(1) | C shell built-in commands |
| **gprof** | **gprof**(1) | display call-graph profile data |
| **graph** | **graph**(1G) | draw a graph |
| **grep** | **grep**(1V) | search a file for a string or regular expression |
| **groups** | **groups**(1) | display a user's group memberships |
| **hashstat** | **csh**(1) | C shell built-in commands |
| **head** | **head**(1) | display first few lines of specified files |
| **help** | **help**(1) | ask for help regarding SCCS errors or warnings |
| **help_viewer** | **help_viewer**(1) | SunView help application |
| **history** | **csh**(1) | C shell built-in commands |
| **hostid** | **hostid**(1) | print the numeric identifier of the current host |
| **hostname** | **hostname**(1) | set or print name of current host system |
| **hpplot** | **plot**(1G) | graphics filters for various plotters |

| | | |
|---|---|---|
| **i386** | **machid**(1) | return true if processor is of a given type |
| **iAPX286** | **machid**(1) | return true if processor is of a given type |
| **iconedit** | **iconedit**(1) | edit images for SunView icons, cursors and panels |
| **id** | **id**(1) | print the user name and ID, and group name and ID |
| **if** | **csh**(1) | C shell built-in commands |
| **implot** | **plot**(1G) | graphics filters for various plotters |
| **indent** | **indent**(1) | indent and format a C program source file |
| **indentpro_to_defaults** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **indxbib** | **indxbib**(1) | create an inverted index to a bibliographic database |
| **inline** | **inline**(1) | in-line procedure call expander |
| **input_from_defaults** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **input_from_defaults** | **input_from_defaults**(1) | update the mouse and keyboard from defaults |
| **insert_brackets** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **install** | **install**(1) | install files |
| **ipcrm** | **ipcrm**(1) | remove message queue, semaphore, shared memory ID |
| **ipcs** | **ipcs**(1) | report interprocess communication facilities status |
| **jobs** | **csh**(1) | C shell built-in commands |
| **join** | **join**(1) | relational database operator |
| **keylogin** | **keylogin**(1) | decrypt and store secret key |
| **kill** | **kill**(1) | send a signal to a process, or terminate a process |
| **label** | **csh**(1) | C shell built-in commands |
| **last** | **last**(1) | indicate last logins by user or terminal |
| **lastcomm** | **lastcomm**(1) | show the last commands executed, in reverse order |
| **ld.so** | **ld**(1) | link editor, dynamic link editor |
| **ld** | **ld**(1) | link editor, dynamic link editor |
| **ldd** | **ldd**(1) | list dynamic dependencies |
| **leave** | **leave**(1) | remind you when you have to leave |
| **lex** | **lex**(1) | lexical analysis program generator |
| **limit** | **csh**(1) | C shell built-in commands |
| **line** | **line**(1) | read one line |
| **lint** | **lint**(1V) | a C program verifier |
| **ln** | **ln**(1) | make hard or symbolic links to files |
| **load** | **load**(1) | load Sun386i clusters |
| **loadc** | **load**(1) | load Sun386i clusters |
| **lockscreen** | **lockscreen**(1) | maintain SunView context and prevent unauthorized access |
| **lockscreen_default** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **lockscreen_default** | **lockscreen**(1) | maintain SunView context and prevent unauthorized access |
| **logger** | **logger**(1) | add entries to the system log |
| **login** | **login**(1) | log in to the system |
| **logname** | **logname**(1) | get the name by which you logged in |
| **logout** | **csh**(1) | C shell built-in commands |
| **look** | **look**(1) | find words in the system dictionary or lines in a sorted list |
| **lookbib** | **lookbib**(1) | find references in a bibliographic database |
| **lorder** | **lorder**(1) | find an ordering relation for an object library |
| **lpq** | **lpq**(1) | display the queue of printer jobs |
| **lpr** | **lpr**(1) | send a job to the printer |
| **lprm** | **lprm**(1) | remove jobs from the printer queue |
| **lptest** | **lptest**(1) | generate lineprinter ripple pattern |
| **ls** | **ls**(1V) | list the contents of a directory |
| **m4** | **m4**(1V) | macro language processor |
| **m68k** | **machid**(1) | return true if processor is of a given type |
| **mach** | **mach**(1) | display the processor type of the current host |
| **machid** | **machid**(1) | return true if processor is of a given type |

| | | |
|---|---|---|
| **mail** | **mail**(1) | read or send mail messages |
| **mailrc_to_defaults** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **mailtool** | **mailtool**(1) | SunView interface for the mail program |
| **make** | **make**(1) | maintain, update, and regenerate related programs and files |
| **man** | **man**(1) | display reference manual pages; find pages by keyword |
| **mesg** | **mesg**(1) | permit or deny messages on the terminal |
| **mkdir** | **mkdir**(1) | make a directory |
| **mkstr** | **mkstr**(1) | create an error message file by massaging C source files |
| **more** | **more**(1) | browse or page through a text file |
| **mt** | **mt**(1) | magnetic tape control |
| **mv** | **mv**(1) | move or rename files |
| **neqn** | **eqn**(1) | typeset mathematics |
| **newgrp** | **newgrp**(1) | log in to a new group |
| **nice** | **nice**(1) | run a command at low priority |
| **nl** | **nl**(1) | line numbering filter |
| **nm** | **nm**(1) | print name list |
| **nohup** | **nohup**(1V) | run a command immune to hangups and quits |
| **notify** | **csh**(1) | C shell built-in commands |
| **nroff** | **nroff**(1) | format documents for display or line-printer |
| **objdump** | **objdump**(1) | dump selected parts of a COFF object file |
| **od** | **od**(1V) | octal, decimal, hex, and ascii dump |
| **oldccat** | **oldcompact**(1) | compress and uncompress files, and cat them |
| **oldcompact** | **oldcompact**(1) | compress and uncompress files, and cat them |
| **oldeyacc** | **oldeyacc**(1) | modified yacc allowing much improved error recovery |
| **oldfilemerge** | **oldfilemerge**(1) | window-based file comparison and merging program |
| **oldmake** | **oldmake**(1) | maintain, update, and regenerate groups of programs |
| **oldprmail** | **oldprmail**(1) | display waiting mail |
| **oldpti** | **oldpti**(1) | phototypesetter interpreter |
| **oldsetkeys** | **oldsetkeys**(1) | modify interpretation of the keyboard |
| **oldsun3cvt** | **oldsun3cvt**(1) | convert Sun-2 executables for use on a Sun-3 |
| **oldsyslog** | **oldsyslog**(1) | make a system log entry |
| **oldtektool** | **oldtektool**(1) | SunView Tektronix 4014 terminal-emulator window |
| **olduncompact** | **oldcompact**(1) | compress and uncompress files, and cat them |
| **oldvc** | **oldvc**(1) | version control |
| **on** | **on**(1C) | execute on remote system with local environment |
| **onintr** | **csh**(1) | C shell built-in commands |
| **organizer** | **organizer**(1) | file and directory manager |
| **overview** | **overview**(1) | run a program from SunView that takes over the screen |
| **pack** | **pack**(1) | compress and expand files |
| **page** | **more**(1) | browse or page through a text file |
| **pagesize** | **pagesize**(1) | display the size of a page of memory |
| **passwd** | **passwd**(1) | change password file information |
| **paste** | **paste**(1) | join lines of several files |
| **pcat** | **pack**(1) | compress and expand files |
| **pdp11** | **machid**(1) | return true if processor is of a given type |
| **perfmeter** | **perfmeter**(1) | display system performance values in a meter or strip chart |
| **pg** | **pg**(1V) | page through a file on a soft-copy terminal |
| **plot** | **plot**(1G) | graphics filters for various plotters |
| **popd** | **csh**(1) | C shell built-in commands |
| **pr** | **pr**(1V) | prepare file(s) for printing, perhaps in multiple columns |
| **printenv** | **printenv**(1) | display environment variables currently set |
| **prof** | **prof**(1) | display profile data |
| **prs** | **prs**(1) | display selected portions an SCCS history |

| | | |
|---|---|---|
| **prt** | **prt**(1) | display the delta and commentary history of an SCCS file |
| **ps** | **ps**(1) | display the status of current processes |
| **ptx** | **ptx**(1) | generate a permuted index |
| **pushd** | **csh**(1) | C shell built-in commands |
| **pwd** | **pwd**(1) | display the pathname of the current working directory |
| **quota** | **quota**(1) | display a user's disk quota and usage |
| **ranlib** | **ranlib**(1) | convert archives to random libraries |
| **rasfilter8to1** | **rasfilter8to1**(1) | convert an 8-bit deep rasterfile to a 1-bit deep rasterfile |
| **rastrepl** | **rastrepl**(1) | magnify a raster image by a factor of two |
| **rcp** | **rcp**(1C) | remote file copy |
| **rdist** | **rdist**(1) | remote file distribution program |
| **refer** | **refer**(1) | expand and insert references from a bibliographic database |
| **rehash** | **csh**(1) | C shell built-in commands |
| **repeat** | **csh**(1) | C shell built-in commands |
| **reset** | **tset**(1) | establish or restore terminal characteristics |
| **rev** | **rev**(1) | reverse the order of characters in each line |
| **rlogin** | **rlogin**(1C) | remote login |
| **rm** | **rm**(1) | remove (unlink) files or directories |
| **rmdel** | **rmdel**(1) | remove a delta from an SCCS file |
| **rmdir** | **rm**(1) | remove (unlink) files or directories |
| **roffbib** | **roffbib**(1) | format and print a bibliographic database |
| **rpcgen** | **rpcgen**(1) | an RPC protocol compiler |
| **rsh** | **rsh**(1C) | remote shell |
| **rup** | **rup**(1C) | show host status of local machines (RPC version) |
| **ruptime** | **ruptime**(1C) | show host status of local machines |
| **rusers** | **rusers**(1C) | who's logged in on local machines (RPC version) |
| **rwall** | **rwall**(1C) | write to all users over a network |
| **rwho** | **rwho**(1C) | who's logged in on local machines |
| **sact** | **sact**(1) | print current SCCS file editing activity |
| **sccs** | **sccs**(1) | front end for the Source Code Control System (SCCS) |
| **sccsdiff** | **sccsdiff**(1) | compare two versions of an SCCS file |
| **screenblank** | **screenblank**(1) | turn off the screen when the mouse and keyboard are idle |
| **screendump** | **screendump**(1) | dump a frame-buffer image to a file |
| **screenload** | **screenload**(1) | load a frame-buffer image from a file |
| **script** | **script**(1) | make typescript of a terminal session |
| **scrolldefaults** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **sdiff** | **sdiff**(1) | contrast two text files by displaying them side-by-side |
| **sed** | **sed**(1V) | stream editor |
| **selection_svc** | **selection_svc**(1) | SunView selection service |
| **set** | **csh**(1) | C shell built-in commands |
| **setenv** | **csh**(1) | C shell built-in commands |
| **sh** | **sh**(1) | the standard UNIX system shell |
| **shelltool** | **shelltool**(1) | run a shell (or command) in a SunView terminal window |
| **shift** | **csh**(1) | C shell built-in commands |
| **shift_lines** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **size** | **size**(1) | display the size of an object file |
| **sleep** | **sleep**(1) | suspend execution for a specified interval |
| **snap** | **snap**(1) | SunView application for system and network administration |
| **soelim** | **soelim**(1) | resolve and eliminate .so requests from nroff or troff input |
| **sort** | **sort**(1V) | sort and collate lines |
| **sortbib** | **sortbib**(1) | sort a bibliographic database |
| **source** | **csh**(1) | C shell built-in commands |
| **sparc** | **machid**(1) | return true if processor is of a given type |

| | | |
|---|---|---|
| **spell** | **spell**(1) | report spelling errors |
| **spellin** | **spell**(1) | report spelling errors |
| **spellout** | **spell**(1) | report spelling errors |
| **spline** | **spline**(1G) | interpolate smooth curve |
| **split** | **split**(1) | split a file into pieces |
| **stop** | **csh**(1) | C shell built-in commands |
| **strings** | **strings**(1) | find printable strings in an object file or binary |
| **strip** | **strip**(1) | remove symbols and relocation bits from an object file |
| **stty** | **stty**(1V) | set or alter the options for a terminal |
| **stty_from_defaults** | **defaultsedit**(1) | create or edit default settings for SunView 1 |
| **stty_from_defaults** | **stty_from_defaults**(1) | set terminal editing characters from the defaults database |
| **su** | **su**(1) | super-user, temporarily switch to a new user ID |
| **sum** | **sum**(1V) | calculate a checksum for a file |
| **sun** | **machid**(1) | return true if processor is of a given type |
| **sunview** | **sunview**(1) | the SunView window environment |
| **suspend** | **csh**(1) | C shell built-in commands |
| **swin** | **swin**(1) | set or get SunView user input options |
| **switch** | **csh**(1) | C shell built-in commands |
| **switcher** | **switcher**(1) | switch between multiple SunView desktops |
| **symorder** | **symorder**(1) | rearrange a list of symbols |
| **sync** | **sync**(1) | update the super block; force changed blocks to the disk |
| **sysex** | **sysex**(1) | invoke the system exerciser |
| **syswait** | **syswait**(1) | execute a command, suspending termination until user input |
| **t300** | **plot**(1G) | graphics filters for various plotters |
| **t300s** | **plot**(1G) | graphics filters for various plotters |
| **t4013** | **plot**(1G) | graphics filters for various plotters |
| **t450** | **plot**(1G) | graphics filters for various plotters |
| **tabs** | **tabs**(1V) | set tab stops on a terminal |
| **tail** | **tail**(1) | display the last part of a file |
| **talk** | **talk**(1) | talk to another user |
| **tar** | **tar**(1) | create tape archives, and add or extract files |
| **tbl** | **tbl**(1) | format tables for nroff or troff |
| **tcopy** | **tcopy**(1) | copy a magnetic tape |
| **tcov** | **tcov**(1) | construct test coverage analysis and statement profile |
| **tee** | **tee**(1) | replicate the standard output |
| **tek** | **plot**(1G) | graphics filters for various plotters |
| **telnet** | **telnet**(1C) | interface to remote system using TELNET protocol |
| **test** | **test**(1V) | return true or false according to a conditional expression |
| **textedit** | **textedit**(1) | SunView window- and mouse-based text editor |
| **textedit_filters** | **textedit_filters**(1) | filters provided with **textedit**(1) |
| **tftp** | **tftp**(1C) | trivial file transfer program |
| **then** | **csh**(1) | C shell built-in commands |
| **time** | **time**(1V) | time a command |
| **tip** | **tip**(1C) | terminal emulator, telephone connection to a remote system |
| **toolplaces** | **toolplaces**(1) | display SunView window locations |
| **touch** | **touch**(1V) | update the access and modification times of a file |
| **tput** | **tput**(1V) | initialize a terminal or query the terminfo database |
| **tr** | **tr**(1V) | translate characters |
| **trace** | **trace**(1) | trace system calls and signals |
| **traffic** | **traffic**(1C) | SunView program to display Ethernet traffic |
| **troff** | **troff**(1) | typeset or format documents |
| **true** | **true**(1) | provide truth values |
| **tset** | **tset**(1) | establish or restore terminal characteristics |

| | | |
|---|---|---|
| **tsort** | **tsort**(1) | topological sort |
| **tty** | **tty**(1) | display the name of the terminal |
| **u3b15** | **machid**(1) | return true if processor is of a given type |
| **u3b2** | **machid**(1) | return true if processor is of a given type |
| **u3b5** | **machid**(1) | return true if processor is of a given type |
| **u3b** | **machid**(1) | return true if processor is of a given type |
| **ul** | **ul**(1) | do underlining |
| **umask** | **csh**(1) | C shell built-in commands |
| **unalias** | **csh**(1) | C shell built-in commands |
| **uname** | **uname**(1V) | display the name of the current system |
| **uncompress** | **compress**(1) | compress or expand files, display expanded contents |
| **unexpand** | **expand**(1) | expand TAB characters to SPACE characters |
| **unget** | **unget**(1) | undo a previous get of an SCCS file |
| **unhash** | **csh**(1) | C shell built-in commands |
| **unifdef** | **unifdef**(1) | resolve and remove ifdef'ed lines from cpp input |
| **uniq** | **uniq**(1) | remove or report adjacent duplicate lines |
| **units** | **units**(1) | conversion program |
| **unix2dos** | **unix2dos**(1) | convert text file from SunOS format to DOS format |
| **unlimit** | **csh**(1) | C shell built-in commands |
| **unload** | **unload**(1) | unload Sun386i clusters |
| **unloadc** | **unload**(1) | unload Sun386i clusters |
| **unpack** | **pack**(1) | compress and expand files |
| **unset** | **csh**(1) | C shell built-in commands |
| **unsetenv** | **csh**(1) | C shell built-in commands |
| **uptime** | **uptime**(1) | show how long the system has been up |
| **users** | **users**(1) | display a compact list of users logged in |
| **uucp** | **uucp**(1C) | system to system copy |
| **uudecode** | **uuencode**(1C) | encode a binary file, or decode its ASCII representation |
| **uuencode** | **uuencode**(1C) | encode a binary file, or decode its ASCII representation |
| **uulog** | **uucp**(1C) | system to system copy |
| **uuname** | **uucp**(1C) | system to system copy |
| **uusend** | **uusend**(1C) | send a file to a remote host |
| **uustat** | **uustat**(1C) | uucp status inquiry and job control |
| **uux** | **uux**(1C) | remote system command execution |
| **vacation** | **vacation**(1) | reply to mail automatically |
| **val** | **val**(1) | validate an SCCS file |
| **vax** | **machid**(1) | return true if processor is of a given type |
| **vfontinfo** | **vfontinfo**(1) | inspect and print out information about fonts |
| **vgrind** | **vgrind**(1) | grind nice program listings |
| **vi** | **vi**(1) | visual display editor based on **ex**(1) |
| **view** | **vi**(1) | visual display editor based on **ex**(1) |
| **vplot** | **vplot**(1) | plot graphics for a Versatec printer |
| **vswap** | **vswap**(1) | convert a foreign font file |
| **vtroff** | **vtroff**(1) | troff to a raster plotter |
| **vwidth** | **vwidth**(1) | make a troff width table for a font |
| **w** | **w**(1) | who is logged in, and what are they doing |
| **wait** | **wait**(1) | wait for a process to finish |
| **wall** | **wall**(1) | write to all users logged in |
| **wc** | **wc**(1) | display a count of lines, words and characters |
| **what** | **what**(1) | identify the version of files under SCCS |
| **whatis** | **whatis**(1) | display a one-line summary about a keyword |
| **whereis** | **whereis**(1) | locate binary, source, and manual page for a command |
| **which** | **which**(1) | locate a command; display its pathname or alias |

| | | |
|---|---|---|
| **while** | **csh**(1) | C shell built-in commands |
| **who** | **who**(1) | who is logged in on the system |
| **whoami** | **whoami**(1) | display the effective current username |
| **whois** | **whois**(1) | DARPA Internet user name directory service |
| **write** | **write**(1) | write a message to another user |
| **xargs** | **xargs**(1) | construct the arguments list(s) and execute a command |
| **xget** | **xsend**(1) | send or receive secret mail |
| **xsend** | **xsend**(1) | send or receive secret mail |
| **xstr** | **xstr**(1) | extract strings from C programs to implement shared strings |
| **yacc** | **yacc**(1) | yet another compiler-compiler: parsing program generator |
| **yes** | **yes**(1) | be repetitively affirmative |
| **ypcat** | **ypcat**(1) | print values in a YP data base |
| **ypmatch** | **ypmatch**(1) | print the value of one or more keys from a YP map |
| **yppasswd** | **yppasswd**(1) | change your network password in the Yellow Pages |
| **ypwhich** | **ypwhich**(1) | which host is the YP server or map master? |
| **zcat** | **compress**(1) | compress or expand files, display expanded contents |

NAME

adb – general-purpose debugger

SYNOPSIS

**adb** [ **–w** ] [ **–k** ] [ **–I** *dir* ] [ *objectfile* [ *corefile* ] ]

AVAILABILITY

This command is available with the *Debugging* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**adb** is an interactive, general-purpose debugger. It can be used to examine files and provides a controlled environment for the execution of programs.

*objectfile* is normally an executable program file, preferably containing a symbol table. If the file does not contain a symbol table, it can still be examined, but the symbolic features of **adb** cannot be used. The default for *objectfile* is **a.out**. *corefile* is assumed to be a core image file produced after executing *objectfile*. The default for *corefile* is **core**.

OPTIONS

**–k** Perform kernel memory mapping; should be used when *corefile* is a system crash dump or **/dev/mem**.

**–w** Create both *objectfile* and *corefile*, if necessary, and open them for reading and writing so that they can be modified using **adb**.

**–I** *dir* specifies a directory where files to be read with **$<** or **$<<** (see below) will be sought; the default is **/usr/lib/adb**.

USAGE

Refer to **adb** in *Debugging Tools* for more complete information on how to use **adb**. (Note: Some commands require that you compile progams to be debugged with the **–go** compiler flag; see **cc**(1V) for details. These commands are not currently available on Sun-4 systems; they are marked '(**–go** only)' below.)

**Commands**

**adb** reads commands from the standard input and displays responses on the standard output. It does not supply a prompt. It ignores the QUIT signal. INTERRUPT invokes the next **adb** command. **adb** generally recognizes command input of the form:

[ *address* ] [, *count* ] *command* [ ; ]

*address* and *count* (if supplied) are expressions that result, respectively, in a new current address, and a repetition count. *command* is composed of a verb followed by a modifier or list of modifiers.

The symbol '**.**' represents the current location. It is initially zero. The default *count* is '**1**'.

*Verbs*

| | |
|---|---|
| **?** | Print locations starting at *address* in *objectfile*. |
| **/** | Print locations starting at *address* in *corefile*. |
| **=** | Print the value of *address* itself. |
| **@** | Interpret *address* as a source address. Print locations in *objectfile* or lines of source, as appropriate. (**–go** only). |
| **:** | Manage a subprocess. |
| **$r** | Print names and contents of CPU registers. |
| **$R** | Print names and contents of MC68881 registers, if any. |
| **$x** | Print the names and contents of FPA registers 0 through 15, if any. |
| **$X** | Print the names and contents of FPA registers 16 through 31, if any. |
| **>** | Assign a value to a variable or register. |
| RETURN | Repeat the previous command with a *count* of 1. Increment '**.**'. |
| **!** | Shell escape. |

**Modifiers**

Modifiers specify the format of command output. Each modifier consists of a letter, preceded by an integer repeat count.

*Format Modifiers*

The following format modifiers apply to the commands **?**, **/**, **@**, and **=**. To specify a format, follow the command with an optional repeat count, and the desired format letter or letters:

[ *v* ][ [ *r* ]*f*... ]

where *v* is one of these four command verbs, *r* is a repeat count, and *f* is one of the format letters listed below:

| | |
|---|---|
| **o** | ('.' increment: 2) Print 2 bytes in octal. |
| **O** | (4) Print 4 bytes in octal. |
| **q** | (2) Print in signed octal. |
| **Q** | (4) Print long signed octal. |
| **d** | (2) Print in decimal. |
| **D** | (4) Print long decimal. |
| **x** | (2) Print 2 bytes in hexadecimal. |
| **X** | (4) Print 4 bytes in hexadecimal. |
| **h** | (2) Sun386i systems only. Print 2 bytes in hexadecimal in reverse order. |
| **H** | (4) Sun386i systems only. Print 4 bytes in hexadecimal in reverse order. |
| **u** | (2) Print as an unsigned decimal number. |
| **U** | (4) Print long unsigned decimal. |
| **f** | (4) Print a single-precision floating-point number. |
| **F** | (8) Print a double-precision floating-point number. |
| **e** | or **E** (12) Print a 96-bit MC68881 extended-precision floating-point number. (Sun-2 or Sun-3 systems only.) |
| **b** | (1) Print the addressed byte in octal. |
| **B** | (1) Sun386i systems only. Print the addressed byte in hexadecimal. |
| **c** | (1) Print the addressed character. |
| **C** | (1) Print the addressed character using ^ escape convention. |
| **s** | (*n*) Print the addressed string. |
| **S** | (*n*) Print a string using the ^ escape convention. |
| **Y** | (4) Print 4 bytes in date format. |
| **i** | (*n*) Print as machine instructions. |
| **M** | (*n*) On Sun386i systems, print as machine instructions along with machine code. |
| **z** | (*n*) Print with MC68010 machine instruction timings. (Sun-2 or Sun-3 system only.) |
| **I** | (0) Print the source text line specified by '.' (–go only) |
| **a** | (0) Print the value of '.' in symbolic form. |
| **p** | (4) Print the addressed value in symbolic form. |
| **A** | (0) Print the value of '.' in source-symbol form. |
| **P** | (4) Print the addressed value in source-symbol form. |
| **t** | (0) Tab to the next appropriate TAB stop. |
| **r** | (0) Print a SPACE. |
| **n** | (0) Print a NEWLINE. |
| **'...'** | (0) Print the enclosed string. |
| **^** | (0) Decrement '.'. |
| **+** | (0) Increment '.'. |
| **–** | (0) Decrement '.' by 1. |

*Modifiers for ? and / Only*

| | |
|---|---|
| **l** *value mask* | Apply *mask* and compare for *value*; move '.' to matching location. |
| **L** *value mask* | Apply *mask* and compare for 4-byte *value*; move '.' to matching location. |
| **w** *value* | Write the 2-byte *value* to address. |
| **W** *value* | Write the 4-byte *value* to address. |

**m** *b1 e1 f1*[ ?] Map new values for *b1, e1, f1*. If the **?** or **/** is followed by ***** then the second segment (*b2 , e2 , f2*) of the address mapping is changed.

: *Modifiers*

**b** *commands* Set breakpoint, execute *commands* when reached.

**B** *commands* Set breakpoint using source address, execute *commands* when reached (**–go** only).

**w** *commands* Sun386i systems only. Set a data write breakpoint at *address* . Like **b** except that the breakpoint is hit when the program writes to *address* .

**D** Delete breakpoint at source address (**–go** only).

**r** Run *objectfile* as a subprocess.

**c***s* The subprocess is continued with signal *s*.

**s***s* Single-step the subprocess with signal *s*.

**S***s* Single-step the subprocess with signal *s* using source lines (**–go** only).

**i** Add the signal specified by *address* to the list of signals passed directly to the subprocess.

**t** Remove the signal specified by *address* from the list implicitly passed to the subprocess.

**k** Single-step the subprocess with signal *s* using source lines ( **–go** only)

**e***s* Sun386i systems only. Like *s* , but steps over subroutine calls instead of into them.

**u** Sun386i systems only. Continue uplevel, stopping after the current routine has returned. Should only be given after the frame pointer has been pushed on the stack.

**i** Add the signal specified by *address* to the list of signals passed directly to the subprocess

**t** Remove the signal specified by *address* from the list implicitly passed to the subprocess.

**k** Terminate the current subprocess, if any.

**A** Sun386i systems only. Attach the process whose process ID is given by *address*. The PID is generally preceded by **0t** so that it will be interpreted in decimal.

**R** Sun386i systems only. Release (detach) the current process.

$ *Modifiers*

*<file* Read commands from the file *file*.

*<<file* Similar to **<**, but can be used in a file of commands without closing the file.

*>file* Append output to *file*, which is created if it does not exist.

**?** Print process ID, the signal which stopped the subprocess, and the registers.

**r** Print the names and contents of the general CPU registers, and the instruction addressed by **pc**.

**R** On Sun-3 systems with an MC68881 floating-point coprocessor, print the names and contents of the coprocessor's registers.

**x** On Sun-3 systems with a Floating Point Accelerator (FPA), print the names and contents of FPA floating-point registers 0 through 15. On Sun-4 systems, print the names and contents of the floating-point registers 0 through 15.

**X** On Sun-3 systems with an FPA, print the names and contents of FPA registers 16 through 31. On Sun-4 systems, print the names and contents of floating-point registers 16 through 31.

**b** Print all breakpoints and their associated counts and commands.

**c** C stack backtrace. On Sun-4 systems, it is impossible for **adb** to determine how many parameters were passed to a function. The default that **adb** chooses in a **$c** command is to show the six parameter registers. This can be overridden by appending a hexadecimal number to the **$c** command, specifying how many parameters to display. For example, the **$cf** command will print 15 parameters for each function in the stack trace.

**C** C stack backtrace with names and (32 bit) values of all automatic and static variables for each active function. (**–go** only)

**d** Set the default radix to *address* and report the new value. Note: *address* is interpreted in the (old) current radix. Thus '**10$d**' never changes the default radix.

**e** Print the names and values of external variables.

**w** Set the page width for output to *address* (default 80).

| | |
|---|---|
| **s** | Set the limit for symbol matches to *address* (default 255). |
| **o** | All integers input are regarded as octal. |
| **q** | Exit from **adb**. |
| **v** | Print all non zero variables in octal. |
| **m** | Print the address map. |
| **f** | Print a list of known source filenames. (**–go** only) |
| **p** | Print a list of known procedure names. (**–go** only) |
| **p** | (*Kernel debugging*) Change the current kernel memory mapping to map the designated **user structure** to the address given by *_u* ( *u* on Sun386i systems); this is the address of the user's **proc** structure. |
| **i** | Show which signals are passed to the subprocess with the minimum of **adb** interference. |
| **W** | Reopen *objectfile* and *corefile* for writing, as though the **–w** command-line argument had been given. |
| **I** | Sun386i systems only. Set the length in bytes (1, 2, or 4) of the object referenced by *:a* and *:w* to *address* Default is 1. |

**Variables**

Named variables are set initially by **adb** but are not used subsequently.

| | |
|---|---|
| **0** | The last value printed. |
| **1** | The last offset part of an instruction source. |
| **2** | The previous value of variable 1. |
| **9** | The count on the last **$<** or **$<<** command. |

On entry the following are set from the system header in the *corefile* or *objectfile* as appropriate.

| | |
|---|---|
| **b** | The base address of the data segment. |
| **B** | On Sun-3 systems, the number of an address register that points to the FPA page. |
| **d** | The data segment size. |
| **e** | The entry point. |
| **F** | On Sun-3 systems, a value of '1' indicates FPA disassembly. |
| **m** | The 'magic' number (0407, 0410 or 0413). |
| **s** | The stack segment size. |
| **t** | The text segment size. |

**Expressions**

| | |
|---|---|
| **.** | The value of *dot*. |
| **+** | The value of *dot* incremented by the current increment. |
| **^** | The value of *dot* decremented by the current increment. |
| **&** | The last *address* typed. (In older versions of **adb**, "'" was used.) |
| *integer* | A number. The prefixes **0o** and **0O** indicate octal; **0t** and **0T**, decimal; **0x** and **0X**, hexadecimal (the default). |
| *int.frac* | A floating-point number. |
| *'cccc'* | ASCII value of up to 4 characters. |
| *<name* | The value of *name*, which is either a variable name or a register name. |
| *symbol* | A symbol in the symbol table. An initial '_' will be prepended to *symbol* (on Sun-2, Sun-3, and Sun-4 systems but not Sun386i systems) if needed. |
| *_symbol* | An external symbol (on Sun-2, Sun-3, and Sun-4 systems but not Sun386i systems). |
| *routine.name* | The address of the variable *name* in the specified routine in the symbol table. If *name* is omitted, the address of the most recent stack frame for *routine*. |
| *(exp)* | The value of *exp*. |

*Unary Operators*

| | |
|---|---|
| ******exp* | The contents of location *exp* in *corefile*. |
| *%exp* | The contents of location *exp* in *objectfile* (In older versions of **adb**, '@' was used). |
| *–exp* | Integer negation. |
| *~exp* | Bitwise complement. |
| *#exp* | Logical negation. |

| | |
|---|---|
| ^F*exp* | (CTRL–F) Translate program address to source address. (–**go** only) |
| ^A*exp* | (CTRL–A) Translates source address to program address. (–**go** only) |
| `*name* | (Backquote) Translates procedure name to sourcefile address. (–**go** only) |
| *"file"* | The sourcefile address for the zero-th line of *file*. (–**go** only) |

*Binary Operators*

Binary operators are left associative and have lower precedence than unary operators.

| | |
|---|---|
| + | Integer addition. |
| – | Integer subtraction. |
| * | Integer multiplication. |
| % | Integer division. |
| & | Bitwise conjunction ("AND"). |
| \| | Bitwise disjunction ("OR"). |
| # | *lhs* rounded up to the next multiple of *rhs*. |

FILES

**a.out**
**core**

SEE ALSO

**cc**(1V), **dbx**(1), **kadb**(8S), **ptrace**(2), **a.out**(5), **core**(5)

*Debugging Tools*

DIAGNOSTICS

**adb**, when there is no current command or format, comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

There does not seem to be any way to clear all breakpoints.

**adb** uses the symbolic information in an old and now obsolete format generated by the –**go** flag of **cc**(1V); it should be changed to use the new format generated by –**g**.

Since no shell is invoked to interpret the arguments of the **:r** command, the customary wild-card and variable expansions cannot occur.

Since there is little type-checking on addresses, using a sourcefile address in an inappropriate context may lead to unexpected results.

The **$c***parameter-count* command is a kluge.

## NAME

addbib – create or extend a bibliographic database

## SYNOPSIS

**addbib** [ **–a** ] [ **–p** *promptfile* ] *database*

## AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

When **addbib** starts up, answering **y** to the initial **Instructions?** prompt yields directions; typing **n** or RETURN skips them. **addbib** then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to *database*. A null response (just RETURN) means to leave out that field. A – (minus sign) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating **Continue?** prompt allows the user either to resume by typing **y** or RETURN, to quit the current session by typing **n** or **q**, or to edit *database* with any system editor (**vi**(1), **ex**(1), **ed**(1)).

## OPTIONS

**–a** Suppress prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL–D.

**–p** Use a new prompting skeleton, defined in *promptfile*. This file should contain prompt strings, a TAB, and the key-letters to be written to the *database*.

## USAGE

### Bibliography Key Letters"

The most common key-letters and their meanings are given below. **addbib** insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

**%A** Author's name

**%B** Book containing article referenced

**%C** City (place of publication)

**%D** Date of publication

**%E** Editor of book containing article referenced

**%F** Footnote number or label (supplied by **refer**)

**%G** Government order number

**%H** Header commentary, printed before reference

**%I** Issuer (publisher)

**%J** Journal containing article

**%K** Keywords to use in locating reference

**%L** Label field used by **–k** option of **refer**

**%M** Bell Labs Memorandum (undefined)

**%N** Number within volume

**%O** Other commentary, printed at end of reference

**%P** Page number(s)

**%Q** Corporate or Foreign Author (unreversed)

**%R** Report, paper, or thesis (unpublished)

**%S** Series title

**%T** Title of article or book

**%V** Volume number

**%X** Abstract — used by **roffbib**, not by **refer**

**%Y,Z** Ignored by **refer**

EXAMPLE

Except for A, each field should be given just once. Only relevant fields should be supplied.

**%A** **Mark Twain**
**%T** **Life on the Mississippi**
**%I** **Penquin Books**
**%C** **New York**
**%D** **1978**

SEE ALSO

**ed**(1), **ex**(1), **indxbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1), **vi**(1)

**refer** in *Formatting Documents*

NAME

adjacentscreens – connect multiple screens to SunView window driver

SYNOPSIS

**adjacentscreens** [ **–c** | **–m** ] *center-screen* [ **–l** | **–r** | **–t** | **–b** *side-screen* ] ... **–x**

AVAILABILITY

This command is available for Sun-2, Sun-3 and Sun-4 systems with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**adjacentscreens** tells the window-driver's mouse-pointer tracking mechanism how to move between screens that contain windows. Note that **sunview**(1) must be running on all screens before **adjacentscreens** is used. Once properly notified using **adjacentscreens**, the mouse pointer slides from one screen to another as you move the pointer past the appropriate edge of a screen.

OPTIONS

**–c***center-screen*

**–m***center-screen*

*center-screen* is the name of a frame buffer device, such as **/dev/fb**; all other physical screen-positions are given relative to this reference point. The **–c** or **–m** flag is optional. If omitted, the first argument is taken as *center-screen*. If no further arguments are given, *center-screen* has no neighbors.

**–l** *side-screen*

**–r** *side-screen*

**–t** *side-screen*

**–b** *side-screen*

**–l** *side-screen*

*side-screen* is also a frame buffer device name, such as **/dev/cgone**. The **–l** flag indicates that *side-screen* is to the left of *center-screen*; **–r** indicates that it is to the right. **–t** Indicates that *side-screen* is on top of (above) *center-screen*; **–b** indicates that it is below. Each neighboring screen can be specified as an option on the command line.

**–x** Suppress the normal notification to a *side-screen* pointer tracker that *center-screen* is its only neighbor. This option is useful if you have a large number of screens or want exotic inter-window pointer movements.

EXAMPLE

The following command:

**example% adjacentscreens /dev/fb –r /dev/cgone**

sets up pointer tracking so that the pointer slides from a monochrome screen (**/dev/fb**) to a color screen (**/dev/cgone**) when the pointer moves off the right hand edge of the monochrome display. Similarly, the pointer slides from the color screen to the monochrome screen when the pointer moves off the color screen's left edge.

FILES

**/usr/bin/adjacentscreens**
**/dev/fb**
**/dev/cgone**

SEE ALSO

**sunview**(1), **switcher**(1)

BUGS

Window systems on all screens must be initialized before running **adjacentscreens**.

NAME

admin – create and administer SCCS files

SYNOPSIS

**/usr/sccs/admin** [ **–bhnz** ] [ **–a***login* ] ... [ **–d***flag* [ *flag-val* ] ] ... [ **–e***login* ] ... [ **–f***flag* [ *flag-val* ] ] ...
[ **–i** [ *name* ] ] [ **–l***list* ] [ **–m** [ *mrlist* ] ] [ **–r***release* ] [ **–t** [ *name* ] ] [ **–y** [ *comment* ] ] *filename* ...

DESCRIPTION

**admin** creates new SCCS files and changes parameters of existing ones. Filenames of SCCS files begin with the '**s.**' prefix. A named file is created if it does not exist already, and its parameters are initialized according to the specified options. Any parameter not initialized by an option is assigned a default value. If a named file does exist, parameters are altered according to the specified options, and other parameters are left as is.

If a directory is named, **admin** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (for which the last component of the path name does not begin with '**s.**') and unreadable files are silently ignored. A filename of '–' means the standard input — each line of the standard input is taken as the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

OPTIONS

Options are explained as though only one named file is to be processed, since they apply to each file independently.

**–b** When used with the **–i** flag, this option indicates that the SCCS file is to contain an encoded version of a binary data file. Normally, **admin** treats a file as binary only if it contains NUL or control characters, or does not end with a NEWLINE. However binary files do not fit these criteria; **–b** forces the initial delta to be flagged as binary, otherwise subsequent deltas that *do* fit the criteria cannot be checked in.

**–h** Check the structure of the SCCS file (see **sccsfile**(5)), and compare a newly computed check-sum with one stored in the first line of the SCCS file.

The **–h** option inhibits writing on the file, so that it nullifies the effect of any other options; it is therefore only meaningful when processing existing files.

**–n** Create a new SCCS file.

**–z** Recompute the file check-sum and store it in the first line of the SCCS file. Note: using the **–z** option on a truly corrupted file may prevent its corrupted state from being detected later on.

**–a** *login* Add a *login* name, or a numerical group-ID, to the list of users who can make deltas to the SCCS file. A group-ID is equivalent to specifying all users in the group. Several **–a** options can appear on a single **admin** command line. As many login-, or group-ID, as desired can be on the list simultaneously. If the list of users is empty, anyone may add deltas.

**–d** *flag* Delete the indicated *flag* from an SCCS file. The **–d** option may be specified only for existing SCCS files, and several **–d** options can be supplied on a single **admin** command.

**–e** *login* Erase a *login* name or numerical group-ID from the list of users allowed to make deltas. Several **–e** options may be used on a single **admin** command line.

**–f** *flag* Set the indicated *flag*, and, possibly, a value for that *flag*. Several **–f** options can be supplied on a single **admin** command line. *flag*s and their values appear in the FLAGS section below.

**–i** [*name*]
Initial text: the file *name* contains the text for the new SCCS file. This text constitutes the initial delta (set of checked changes checked in at one time); see the **–r** option for the delta numbering scheme. If *name* is omitted, the text is obtained from the standard input.

Omitting the **–i** option altogether creates an empty SCCS file. You can only create one SCCS file with an '**admin –i**' command. Creating more than one SCCS file with a single **admin** command requires that they be created empty, in which case the **–i** option should be omitted. Note: the **–i** option implies the **–n** option.

**–l** *list* Unlock the specified *list* of releases. See FLAGS below for a description of the **l** flag and the syntax of a *list*.

**–m** [ *mrlist* ]
The list of Modification Request (MR) numbers is inserted into the SCCS file as the rationale for creating an initial delta. A diagnostics message results if the **v** flag is not set or the MR validation fails.

**–r** *release*
Specify the release for the initial delta. **–r** may be used only if the **–i** option is also used. The initial delta is inserted into release 1 if this option is omitted. The level of the initial delta is always 1, and initial deltas are named 1.1 by default.

**–t** [*name*]
Insert descriptive text contained in the file *name*. The descriptive text file name *must* be supplied when creating a new SCCS file (either or both **–n** and **–i** options) and the **–t** option is used. In the case of existing SCCS files: 1) a **–t** option without a file name removes descriptive text (if any) currently in the SCCS file, and 2) a **–t** option with a file name replaces the descriptive text currently in the SCCS file with any text in the named file.

**–y** [ *comment* ]
Insert a *comment* for the initial delta into the SCCS file. If the **–y** option is omitted, a default comment line is inserted in the form:

date and time created *YY/MM/DD s HH:MM:SS* by *login*

The **–y** option is valid only if the **–i** and/or **–n** options are specified.

FLAGS

The following *flags* can appear as arguments to the **–f** (set flags) and **–d** (delete flags) options:

**b** When set, the **–b** option can be used on a **get**(1) command to create branch deltas.

**c** *ceil* The highest release (ceiling) which may be retrieved by a **get**(1) command for editing. The ceiling is a number less than or equal to 9999. The default value for an unspecified **c** flag is 9999.

**f** *floor* The lowest release (floor) which may be retrieved by a **get**(1) command for editing. The floor is a number greater than 0 but less than 9999. The default value for an unspecified **f** flag is 1.

**d** *SID* The default delta number (SID) to be used by a **get**(1) command.

**i** Treats the '**No id keywords (ge6)**' message issued by **get**(1) or **delta**(1) as a fatal error. In the absence of the **i** flag, the message is only a warning. The message is displayed if no SCCS identification keywords (see **get**(1)) are found in the text retrieved or stored in the SCCS file.

**j** Concurrent **get**(1) commands for editing may apply to the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

**l** *list* A *list* of locked releases to which deltas can no longer be made. A '**get –e**' fails when applied against one of these locked releases. The *list* has the following syntax:

*<list>* ::= *<range>* | *<list>* , *<range>*
*<range>* ::= *Release Number* | **a**

The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file.

**n** The **delta**(1) command creates a "null" delta in each release (if any) being skipped when a delta is made in a *new* release. For example, releases 3 and 4 are skipped when making delta 5.1 after delta 2.7. These null deltas serve as "anchor points" so that branch deltas may be created from them later. If the **n** flag is absent from the SCCS file, skipped releases will be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

**q** *text* *text* is defined by the user. The *text* is substituted for all occurrences of the **%Q%** keyword in SCCS file text retrieved by **get**(1).

**m** *module*
*module* name of the SCCS file substituted for all occurrences of the **%M%** keyword in the SCCS file text retrieved by **get**(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

**t** *type* *type* of module in the SCCS file substituted for all occurrences of **%Y%** keyword in SCCS file text retrieved by **get**(1).

**v** [*program*]
Validity checking *program*: **delta**(1) prompts for Modification Request (MR) numbers as the reason for creating a delta. The optional *program* specifies the name of an MR number validity checking *program* (see **delta**(1)). If this flag is set when creating an SCCS file, the **–m** option must also be used even if its value is NULL.

FILES

The last component of all SCCS file names must be of the form **s.***filename*. New SCCS files are given mode 444 (see **chmod**(1V)). All writing done by **admin** is to a temporary file with an '**x**'. prefix, created with mode 444 for a new SCCS file, or with the same mode as an existing SCCS file. After successful execution of **admin**, the existing SCCS file is removed, then replaced with the **x.***filename*. This ensures that changes are made to the SCCS file only when no errors have occurred.

It is recommended that directories containing SCCS files have permission mode 755, and that the SCCS files themselves have mode 444. The mode for directories allows only the owner to modify the SCCS files contained in the directories, while the mode of the SCCS files themselves prevents all modifications except those performed using SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner to allow use of a text editor. However, extreme care must be taken when doing this. The edited file should *always* be processed by an '**admin –h**' to check for corruption, followed by an '**admin –z**' to generate a proper check-sum. Another '**admin –h**' is recommended to ensure that the resulting SCCS file is valid.

**admin** also uses a transient lock file (called **z.***filename*), to prevent simultaneous updates to the SCCS file by different users. See **get**(1) for further information.

SEE ALSO

**chmod**(1V), **delta**(1), **ed**(1), **get**(1), **help**(1), **prs**(1), **sccs**(1), **what**(1), **sccsfile(5)**

*Programming Utilities and Libraries.*

DIAGNOSTICS

Use **help**(1) for explanations.

NAME

ar – create library archives, and add or extract files

SYNOPSIS

**ar d|m|p|q|r|t|x [[ abi** *position-name* ] [ **cilouv** ]] *archive* [ *member-file*... ]

SYSTEM V SYNOPSIS

**ar** [–]**d|m|p|q|r|t|x [[ abi** *position-name* ] [ **cilouvs** ]] *archive* [ *member-file*... ]

DESCRIPTION

**ar** maintains groups of files combined into a single archive file. An archive file comprises a set of member files and header information for each file. The archive header and the headers for the file consist of printable characters (assuming that the characters in the names of the files in the archive are printable), and are in a format portable across all machines. This format is described in detail in **ar**(5)). If an archive is composed of printable files, with printable file names, the entire archive is printable.

**ar** is normally used to create and update library files used by the link editor **ld**(1), but can be used for any similar purpose.

*archive* is the name of the archive file. *member-file* is a member file contained in the archive. If this argument is omitted, the command applies to all entries in the archive. Member names have a maximum of 15 characters, except on Sun386i systems, where they have a maximum of 14 characters. Names longer than this are truncated.

SYSTEM V DESCRIPTION

**ar** will run **ranlib** after modifying an archive, so that the symbol table member of the archive will be kept up-to-date.

OPTIONS

You must indicate only one of: **d**, **m**, **p**, **q**, **r**, **t**, or **x**, which may be followed by one or more *Modifiers*.

**d** Delete the named files from the archive file.

**m** Move the named files to the end of the archive.

**p** Print. If no names are given, all files in the archive are written to the standard output; if no names are given, only those files are written, and they are written in the order that they appear in the archive.

**q** Quick append. Append the named files to the end of the archive file without searching the archive for duplicate names. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece. If this option is used to add a member to an archive, and a member with the same name as that member already exists in the archive, the old member will not be removed; two members with the same name will exist in the archive.

**r** Replace the named files in the archive.

**t** Table of contents. If no names are given, all files in the archive are listed; if names are given, only those files are listed.

**x** Extract. If no names are given, all files in the archive are extracted into the current directory; if names are given, only those files are extracted. In neither case does **x** alter the archive file.

Modifiers

**a** Place new files after *posname* (*posname* argument must be present). Applies only to the **r** and **m** options.

**b** Place new files before *posname* (*posname* argument must be present). Applies only to the **r** and **m** options.

**c** Create. Suppress the message that is produced by default when *archive* is created.

**i** Identical to the **b** modifier.

**l** Local. Place temporary files in the current working directory rather than in the default temporary

directory, **/tmp**.

**o** Old date. When files are extracted with the **x** option, set the "last modified" date to the date recorded in the archive.

**u** Update. Replace only those files that have changed since they were put in the archive. Used with the **r** option.

**v** Verbose. When used with the **r**, **d**, **m**, or **q** option, give a file-by-file description of the creation of a new archive file from the old archive and the constituent files. When used with **x**, give a file-by-file description of the extraction of archive files. When used with **t**, give a long listing of information about the files. When used with **p**, write each member's name to the standard output before writing the member to the standard output.

## SYSTEM V OPTIONS

The options may be preceded by –.

### Modifiers

**s** Force the regeneration of the archive symbol table even if **ar** is not invoked with a command that will modify the archive contents.

## EXAMPLES

Creating a new archive:

```
hermes% ar rcv archive file.o
a - file.o
```

Adding to an archive:

```
hermes% ar rav file.o archive next.c
a - next.c
```

Extracting from an archive:

```
hermes% ar xv archive file.o
x - file.o
hermes% ls file.o
file.o
```

Seeing the table of contents:

```
hermes% ar t archive
file.o
next.c
```

## FILES

| | |
|---|---|
| **/tmp/v*.** | temporaries |
| **/tmp** | |

## SEE ALSO

**ld**(1), **lorder**(1), **ranlib**(1), **ar**(5)

## BUGS

If the same file is mentioned twice in an argument list, it is put in the archive twice.

The "last-modified" date of a file will not be altered by the **o** option unless the user is either the owner of the extracted file or the super-user.

NAME

arch – display the architecture of the current host

SYNOPSIS

**arch**

DESCRIPTION

The **arch** command displays the architecture of the current host system.

SEE ALSO

**mach**(1), **machid**(1)

## NAME

as – Sun-1, Sun-2 and Sun-3, Sun-4 and Sun386i assemblers

## SUN-1, SUN-2 and SUN-3 SYNOPSIS

**as** [ **–L** ] [ **–R** ] [ **–o** *objfile* ] [ **–d2** ] [ **–h** ] [ **–j** ] [ **–J** ] [ **–O** ] [ **–mc68010** ] [ **–mc68020** ] *filename*

## SUN-4 SYNOPSIS

**as** [ **–L** ] [ **–R** ] [ **–o** *objfile* ] [ **–O**[*n*] ] [ **–P** [ [ **–I***path* ] [ **–D***name* ] [ **–D***name=def* ] [ **–U***name* ] ] ... ]
[ **–S**[**C**] ] *filename* ...

## Sun386i SYNOPSIS

**as** [ **–k** ] [ **–o** *objfile* ] [ **–R** ] [ **–V** ] [ **–i386** ]

## DESCRIPTION

**as** translates the assembly source file, *filename* into an executable object file, *objfile*. The Sun-4 assembler recognizes the filename argument '–' as the standard input.

All undefined symbols in the assembly are treated as global.

The output of the assembly is left in the file *objfile*.

## OPTIONS

The following options are common to all Sun architectures. Options for specific Sun architectures are listed below.

**–L** Save defined labels beginning with an **L**, which are normally discarded to save space in the resultant symbol table. The compilers generate many such temporary labels.

**–R** Make the initialized data segment read-only by concatenating it to the text segment. This eliminates the need to run editor scripts on assembly code to make initialized data read-only and shared.

**–o** *objfile*
The next argument is taken as the name of the object file to be produced. If the **–o** flag is not used, the object file is named **a.out**.

### Sun-1, Sun-2 and Sun-3 Options

**–d2** Instruction offsets that involve forward or external references, and with unspecified size, are two bytes long. (See also the **–j** option.)

**–h** Suppress span-dependent instruction calculations. Restrict branches to medium length. Force calls to take the most general form. This option is used when the assembly must be minimized, even at the expense of program size and run-time performance. It results in a smaller and faster program than one produced by the **–J** option, but some very large programs may be unable to use it due to the limits of medium-length branches.

**–j** Use short (pc-relative) branches to resolve jump and jump-to-subroutine instructions to external routines. This is for compact programs for which the **-d2** option is inappropriate due to large-program relocation.

**–J** Suppress span-dependent instruction calculations and force branches and calls to take the most general form. This is useful when assembly time must be minimized, even at the expense of program size and run-time performance.

**–O** Perform span-dependent instruction resolution over entire files rather than just individual procedures.

### Sun-4 Options

**–O**[*n*] Enable peephole optimization corresponding to optimization level *n* (1 if *n* not specified) of the Sun high-level language compilers. This option can be used safely only when assembling code produced by a Sun compiler.

**–P** Run **cpp**(1), the C preprocessor, on the files being assembled. The preprocessor is run separately on each input file, not on their concatenation. The preprocessor output is passed to the assembler.

**–I***path*
**–D***name*
**–D***name=def*
**–U***name*

When **–P** is in effect, these **cpp**(1) options are passed to the C preprocessor, without interpretation by **as**. Otherwise, they are ignored.

**–S[C]** Produce a disassembly of the emitted code to the standard output. This is most useful in combination with the **–O** option, to review optimized code. Adding the character C to the option prevents comment lines from appearing in the output.

**Sun386i Options**

**–k** Create position independent code. Called by **cc –pic**.

**–V** Write the version number of the assembler being run on the standard error output.

**–i386** Confirm that this output is intended for an 80386 processor.

**FILES**

**/tmp/as*** default temporary file

**SEE ALSO**

**adb**(1), **cpp**(1), **dbx**(1), **ld**(1), **a.out**(5) **coff**(5)

*Sun-4 Assembly Language Reference Manual*
*Assembly Language Manual*

**BUGS**

The Sun Pascal compiler qualifies a nested procedure name by chaining the names of the enclosing procedures. This sometimes results in names long enough to abort the Sun-1/2/3 assembler, which currently limits identifiers to 512 characters (the Sun-4 assembler does not have this restriction).

**Sun386i CAVEATS**

There can be only one forward-reference to a symbol per arithmetic expression.

NAME

at, batch – execute a command or script at a specified time

SYNOPSIS

**at** [ **–csm** ] *time* [ *date* ] [ + *increment* ] [ *script* ]
**at –r** *jobs...*
**at –l** [ *jobs...* ]

**batch** [ **–csm** ] [ *script* ]

DESCRIPTION

**at** and **batch** read commands from standard input to be executed at a later time. **at** allows you to specify when the commands should be executed, while jobs queued with **batch** will execute when system load level permits. *script* is the name of a file to be used as command input for the Bourne shell, **sh**(1), the C shell, **csh**(1), or an arbitrary shell specified by the SHELL environment variable. If *script* is omitted, command input is accepted from the standard input.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, and **umask**(2) are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use **at** if their name appears in the file **/var/spool/cron/at.allow**. If that file does not exist, the file **/var/spool/cron/at.deny** is checked to determine if the user should be denied access to **at**. If neither file exists, only the super-user is allowed to submit a job. If **at.deny** is empty, global usage is permitted. The allow/deny files consist of one user name per line.

The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour*:*minute*. A suffix **am** or **pm** may be appended; otherwise a 24-hour clock time is understood. The suffix **zulu** may be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", **today** and **tomorrow** are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.)

Thus legitimate commands include:

**at 0815am Jan 24**
**at 8:15am Jan 24**
**at now + 1 day**
**at 5 pm Friday**

**at** and **batch** write the job number and schedule time to standard error.

**batch** submits a batch job. It is almost equivalent to '**at now**', but not quite. For one, it goes into a different queue. For another, '**at now**' will respond with the error message '**too late**'.

OPTIONS

**–c** C shell. **csh** is used to execute *script*.

**–s** Standard (Bourne) shell. **sh** is used to execute the job. SHELL environment variable to determine which shell to use.

**–m** Mail. Send mail after the job has been run, even if the job completes successfully.

**–r** *jobs* ... Remove the specified *jobs* previously scheduled by **at** or **batch**. The job numbers are the numbers of the jos given to you previously by the **at** or **batch** commands. You can only

remove your own jobs unless you are the super-user.

**-l** [*jobs* ...]
If *jobs* is specified, print the queue entry for those *jobs*; if *jobs* is not specified, print the queue entries for all jobs for the user.

ENVIRONMENT

If neither file exists, only the super-user is allowed to submit a job. If **at.deny** is empty, global usage is permitted. The allow/deny files consist of one user name per line.

EXAMPLES

Unless a *script* is specified, the **at** and **batch** commands read from standard input the commands to be executed at a later time. **sh** and **csh** provide different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

**batch**
**nroff** *filename* **>** *outfile*
**<control-D>** (hold down 'control' and depress '**D**')

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

**batch <<!**
**nroff** *filename* **2>&1 >** *outfile* **| mail** *loginid*
**!**

To have a job reschedule itself, invoke **at** from within the shell procedure, by including code similar to the following within the shell file:

**at 1900 thursday next week** *shellfile*

FILES

| | |
|---|---|
| **/var/spool/cron** | main cron directory |
| **/var/spool/cron/at.allow** | list of allowed users |
| **/var/spool/cron/at.deny** | list of denied users |
| **/var/spool/cron/atjobs** | spool area |

SEE ALSO

**atq**(1), **atrm**(1), **csh**(1), **kill**(1), **mail**(1), **nice**(1), ps(1), **sh**(1), **umask**(2), **cron**(8)

DIAGNOSTICS

Complains about various syntax errors and times out of range.

BUGS

If the system crashes, mail stating that the job was not completed is not sent to the user.

Shell interpreter specifiers (such as, **!/bin/csh**) in the beginning of *script* are ignored.

NAME

atq – display the queue of jobs to be run at specified times

SYNOPSIS

**atq** [ **–c** ] [ **–n** ] *username*...

DESCRIPTION

**atq** prints the queue of jobs, created with the **at**(1) command, that are waiting to be run at later date.

With no flags, the queue is sorted in chronological order of execution.

If no usernames are specified, the entire queue is displayed; otherwise, only those jobs belonging to the named users are displayed.

OPTIONS

**–c** By creation time. Sorted the queue by the time that the **at** command was given, the most recently created job first.

**–n** Number of jobs. Print the total number of jobs currently in the queue. Do not list them.

FILES

**/var/spool/cron** spool area

SEE ALSO

**at(1), atrm(1), cron(8)**

NAME

atrm – remove jobs spooled by at or batch

SYNOPSIS

**atrm** [ **–fi** ] [ – ] [ *job-number* ]... [ *username* ]...

DESCRIPTION

**atrm** removes delayed-execution jobs that were created with the **at**(1) command. The list of jobs can be displayed by **atq**(1)

**atrm** removes each *job-number* you specify, and/or all jobs belonging to *username*, provided that you own the indicated jobs.

Jobs belonging to other users can only be removed by the super-user.

OPTIONS

**–f** Force. All information regarding the removal of the specified jobs is suppressed.

**–i** Interactive. **atrm** asks if a job should be removed; a response of **y** verifies that the job is to be removed.

– Remove jobs that were queued by the current user. If invoked by the super-user, the entire queue will be flushed.

FILES

**/var/spool/cron** spool area

SEE ALSO

**at**(1), **atq**(1), **cron**(8)

NAME

awk – pattern scanning and processing language

SYNOPSIS

**awk** [ **–f** *program-file* ] [ **–F***c* ] [ *program* ] [ *variable* =*value* ... ] [ *filename*... ]

DESCRIPTION

**awk** scans each of its input *filename*s for lines that match any of a set of patterns specified in *program*. The input *filename*s are read in order; the standard input is read if there are no *filename*s. The filename '–' means the standard input.

The set of patterns may either appear literally on the command line as *program*, or, by using the '**–f** *program-file*' option, the set of patterns may be in a *program-file*; a *program-file* of '–' means the standard input. If the *program* is specified on the command line, it should be enclosed in single quotes (') to protect it from the shell.

**awk** variables may be set on the command line using arguments of the form *variable* =*value*. This sets the **awk** variable *variable* to *value* before the first record of the next *filename* argument is read.

With each pattern in *program* there can be an associated action that will be performed when a line of a *filename* matches the pattern. See the discussion below for the format of input lines and the **awk** language. Each line in each input *filename* is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

OPTIONS

**–f** *program-file*
Use the contents of *program-file* as the source for the *program*.

**–F***c* Use the character *c* as the field separator (FS) character. See the discussion of FS below.

USAGE

**Input Lines**

An input line is made up of fields separated by white space. The field separator can be changed by using FS — see **Special Variable Names** below. Fields are denoted **$1**, **$2**, ... , up to **$9**; **$0** refers to the entire line.

**Pattern-action Statements**

A pattern-action statement has the form

*pattern* { *action* }

A missing *action* means copy the line to the output; a missing *pattern* always matches.

**Action Statements**

An *action* is a sequence of *statement*s. A *statement* can be one of the following:

**if** ( *conditional* ) *statement* [ **else** *statement* ]
**while** ( *conditional* ) *statement*
**for** ( *expression* ; *conditional* ; *expression* ) *statement*
**break**
**continue**
{ [ *statement* ] ...}
*variable* = *expression*
**print** [ *expression-list* ] [ > *expression* ]
**printf** *format* [ , *expression-list* ] [ > *expression* ]
**next** skip remaining patterns on this input line
**exit** skip the rest of the input

**Format of the awk Language**

*statement*s are terminated by semicolons, NEWLINE characters or right braces. An empty *expression-list* stands for the whole line.

*expressions* take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The C operators ++ , −− , += , −= , *= , /= , and %= are also available in expressions.

*variable* may be scalars, array elements (denoted *x* [ *i* ]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric, providing a form of associative memory. String constants are quoted "... ".

The **print** statement prints its arguments on the standard output (or on a file if *>filename* is present), separated by the current output field separator, and terminated by the output record separator. The **printf** statement formats its expression list according to the format template *format* (see **printf**(3S) for a description of the formatting control characters).

**Built In Functions**

The built-in function **length** returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions **exp**, **log**, **sqrt**, and **int**, where **int** truncates its argument to an integer. '**substr**( *s*, *m*, *n* )' returns the *n*-character substring of *s* that begins at position *m*. '**sprintf** (*format*, *expression*, *expression*, ...)' formats the expressions according to the **printf** format given by *format*, and returns the resulting string.

**Patterns**

Patterns are arbitrary Boolean combinations (!, || , &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in **egrep** (see **grep**(1)), Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

*expression matchop regular-expression*
*expression relop expression*

where a *relop* is any of the six relational operators in C, and a *matchop* is either ~ (contains) or !~ (does not contain). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special pattern **BEGIN** may be used to capture control before the first input line is read, in which case **BEGIN** must be the first pattern. The special pattern **END** may be used to capture control after the last input line is read, in which case **END** must be the last pattern.

**Special Variable Names**

A single character *c* may be used to separate the fields by starting the program with

**BEGIN {FS = "*c*" }**

or by using the **−F***c* option.

Other variable names with special meanings include **NF**, the number of fields in the current record; **NR**, the ordinal number of the current record; **FILENAME**, the name of the current input file; **OFS**, the output field separator (default blank); **ORS**, the output record separator (default NEWLINE); and **OFMT**, the output format for numbers (default **%.6g**).

**EXAMPLES**

Print lines longer than 72 characters:

**length > 72**

Print first two fields in opposite order:

**{ print $2, $1 }**

Add up first column, print sum and average:

```
{ s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

SEE ALSO

**grep**(1V), **lex**(1), **sed**(1V), **printf**(3S)

*Editing Text Files*

BUGS

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

There is no escape sequence that prints a double-quote. A workaround is to use the **sprintf** (see **printf**(3S)) function to store the character into a variable by its ASCII sequence.

```
dq = sprintf("%c", 34)
```

Syntax errors result in the cryptic message '**awk: bailing out near line 1**'.

## NAME

banner – display a string in large letters

## SYNOPSIS

**/usr/5bin/banner** *strings*

## DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command.

**banner** prints its arguments (each up to 10 characters long) in large letters on the standard output.

## SEE ALSO

**echo**(1V)

NAME

bar – create tape archives, and add or extract files

SYNOPSIS

**bar** [ – ] **crxtu** [ **014578feovwbXlFmhpBisHSUZRTIN** ] [ **barfile** ] [ *blocksize* ] [ *exclude-file* ] [ *string* ] [ *target_directory* ] [ *user_id* ] [ *include-file* ] *filename1* ... [ **–C** *dir filename* ... ]...

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**bar** archives and extracts multiple files onto a single **bar**, file archive, called a *barfile*. It is quite similar to tar(5), but it has additional function modifiers , can read and write multiple volumes, and writes and reads a format that is incompatible with **tar** (see **bar**(5)). A barfile is usually a magnetic tape, but it can be any file. **bar**'s actions are controlled by the first argument, the *key*, a string of characters containing exactly one function letter from the set **rxtuc**, and one or more of the optional function modifiers listed below. Other arguments to **bar** are file or directory names that specify which files to archive or extract. In all cases, the appearance of a directory name refers recursively to the files and subdirectories of that directory.

FUNCTION LETTERS

**c** Create a new **barfile** and write the named files onto it.

**r** Write the named files on the end of the *barfile*. Note: this option *does not work* with quarter-inch archive tapes.

**x** Extract the named files from the *barfile*. If a named file matches a directory with contents written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *filename* arguments are given, all files in the archive are extracted. Note: if multiple entries specifying the same file are on the tape, the last one overwrites all earlier versions.

**t** List the table of contents of the *barfile*.

**u** Add the named files to the **barfile** if they are not there or if they have been modified since they were last archived. Note: this option *does not work* with quarter-inch archive tapes.

FUNCTION MODIFIERS

**014578**

Select an alternate drive on which the tape is mounted. The numbers **2**, **3**, **6**, and **9** do not specify valid drives. The default is **/dev/rmt8**.

**f** Use the next argument as the name of the *barfile*. If **f** is omitted, use the device indicated by the **TAPE** environment variable, if set. Otherwise, use **/dev/rmt8** by default. If *barfile* is given as '–', **bar** writes to the standard output or reads from the standard input, whichever is appropriate. Thus, **bar** can be used as the head or tail of a filter chain. **bar** can also be used to copy hierarchies with the command:

**example% cd fromdir; bar cf – . | (cd todir; bar xfBp –)**

**e** If any unexpected errors occur **bar** will exit immediately with a positive exit status.

**o** Suppress information specifying owner and modes of directories which **bar** normally places in the archive. Such information makes former versions of **bar** generate an error message like:

*<filename>*: **cannot create**

when they encounter it.

**v** Normally **bar** does its work silently; the **v** (verbose) option displays the name of each file **bar** treats, preceded by the function letter. When used with the **t** function, **v** displays the **barfile** entries in a form similar to '**ls –l** '.

**w** Wait for user confirmation before taking the specified action. If you use **w**, **bar** displays the action to be taken followed by the file name, and then waits for a **y** response to proceed. No action is taken on the named file if you type anything other than a line beginning with **y**.

**b** Use the next argument as the blocking factor for tape records. The default blocking factor is 20 blocks. The block size is determined automatically when reading tapes (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to **bar** when it is run. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

**X** Use the next argument as a file containing a list of named files (or directories) to be excluded from the **barfile** when using the key letters '**c**', '**x**', or '**t**'. Multiple **X** arguments may be used, with one *exclude file* per argument.

**l** Display error messages if all links to archived files cannot be resolved. If **l** is not used, no error messages are printed.

**F** With one **F** argument specified, exclude all directories named SCCS from *barfile*. With two arguments **FF**, exclude all directories named SCCS, all files with **.o** as as their suffix, and all files named **errs**, **core**, and **a.out**.

**m** Do not extract modification times of extracted files. The modification time will be the time of extraction.

**h** Follow symbolic links as if they were normal files or directories. Normally, **bar** does not follow symbolic links.

**p** Restore the named files to their original modes, ignoring the present **umask**(2). Setuid and sticky information are also extracted if you are the super-user. This option is only useful with the **x** key letter.

**B** Force **bar** to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that **bar** can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming.

**i** Ignore directory checksum errors.

**s** Force the ownership of extracted files to match the **bar** process's effective user ID and group ID.

**H** The string of up to 128 characters is to be used as a volume header ID. A volume header is written to each volume of the archive when the **c** function letter is specified. See **bar**(5) for the volume header's format.

Use of the **H** function modifier when creating an archive allows **bar** to read volumes out of sequence. When extracting a file that spans volumes, **bar** will identify the tape(s) it needs to extract the entire file. If the wrong volume is inserted, **bar** issues a warning and prompts again for the correct volume.

**S** Place files specified for extraction in this target directory when used with the **x** function letter.

**U** Specify the user ID in the volume header when creating archive, when the **H** function modifier is used. If the **c** function letter is specified and a volume header exists, **bar** will verify that the user ids match before overwriting **barfile** if the **N** modifier is specified.

**Z** Specify compression. **bar** will compress files when used with the **c** function letter and will decompress files when used with the **x** function letter. **bar** will neither compress a compressed file, nor decompress a decompressed file.

**R** Read the volume header and print the information to stdout.

**N** See if the user owns the media (uid matches that in the bar header) before overwriting barfile with the C key word.

**T** When using the **x** or **t** function letters, terminate the search of the media after all the files specified are extracted (for **x**) or listed (for **t**).

**I** Use the next argument as a file containing a list of named files, one per line, to be included in the **bar** archive. The include file expects filenames to be followed by a semicolon and newline character.

In the case where excluded files (see **X** flag) also exist, excluded files take precedence over all included files. So, if a file is specified in both the include and exclude files (or on the command line), it will be excluded.

OPTIONS

–C dir filename

In a **c** (**create**) or **r** (**replace**) operation, **bar** performs a **chdir** (see **csh**(1)) to that directory before interpreting filename. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from **/usr/include** and from **/etc**, one might use:

**example% bar c –C /usr include –C /etc .**

If you get a table of contents from the resulting **barfile**, you will see something like:

**include/**
**include/a.out.h**
*and all the other files in* **/usr/include . . ./chown**
*and all the other files in* **/etc**

Note: the –C option only applies to *one* following directory name and *one* following file name.

EXAMPLES

Here is a simple example using **bar** to create an archive of your home directory on a tape mounted on drive **/dev/rmt0**:

**example% cd**
**example% bar cvf /dev/rmt0 .**
*messages*

The **c** option means create the archive; the **v** option makes **bar** tell you what it's doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt0** in this example).

Here is another example: **/dev/rmt0**:

**example% cd**

**example% bar cvfH /dev/rmt0 "THIS IS MY HEADER" .**
*messages*

As in the first example, the **c** option means create the archive; the **v** option makes **bar** tell you what it's doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt0** in this example). The **H** option says to use the string **"THIS IS MY HEADER"** as the ID field in the volume header.

Now you can read the table of contents from the archive like this:

```
example%        bar tvf /dev/rmt0
(access user-id/group-id        size    mod. date               filename)
rw-r--r-- 1677/40               2123    Nov 7 18:15:1985        ./archive/test.c
...
example%
```

You can extract files from the archive like this:

**example% bar xvf /dev/rmt0**
*messages*

If there are multiple archive files on a tape, each is separated from the following one by an EOF marker. **bar** does not read the EOF mark on the tape after it finishes reading an archive file because **bar** looks for a special header to decide when it has reached the end of the archive. Now if you try to use **bar** to read the

next archive file from the tape, **bar** does not know enough to skip over the EOF mark and tries to read the EOF mark as an archive instead. The result of this is an error message from **bar** to the effect:

**bar: blocksize=0**

This means that to read another archive from the tape, you must skip over the EOF marker before sbarting another **bar** command. You can accomplish this using the *mt* command, as shown in the example below. Assume that you are reading from **/dev/nrmt0**.

**example% bar xvfp /dev/nrmt0** *read first archive from tape*
*messages*
**example% mt fsf 1** *skip over the end-of-file marker*
**example% bar xvfp /dev/nrmt0** *read second archive from tape*
*messages*
**example%**

Finally, here is an example using **bar** to transfer files across the Ethernet. First, here is how to archive files from the local machine (**example**) to a tape on a remote system (**host**):

**example% bar cvfb – 20** *filenames* **|***rsh* **host** *dd*
*messages*
**example%**

In the example above, we are *creating* a **barfile** with the **c** key letter, asking for *verbose* output from **bar** with the **v** option, specifying the name of the output **barfile** using the **f** option (the standard output is where the **barfile** appears, as indicated by the – sign), and specifying the blocksize (20) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the **bar** command *and* on the **dd** command.

Now, here is how to use **bar** to get files from a tape on the remote system back to the local system:

**example% rsh –n host dd if=/dev/rmt0 bs=20b | bar xvBfb – 20** *filenames*
*messages*
**example%**

In the example above, we are *extracting* from the **barfile** with the **x** key letter, asking for *verbose output from* **bar** with the **v** option, telling **bar** it is reading from a pipe with the **B** option, specifying the name of the input **barfile** using the **f** option (the standard input is where the **barfile** appears, as indicated by the '–' sign), and specifying the blocksize (20) with the **b** option.

## FILES

| | |
|---|---|
| **/dev/rmt?** | half-inch magnetic tape interface |
| **/dev/rar?** | quarter-inch magnetic tape interface |
| **/dev/rst?** | SCSI tape interface |
| **/tmp/bar*** | |

## ENVIRONMENT

**TAPE** If specified, in the environment, the value of **TAPE** indicates the default tape device.

## NOTES

**bar** will handle multiple volumes gracefully. If a tape error is encountered, **bar** issues a message on the standard error requesting a new volume. The presence of a new volume is confirmed when **bar** reads a line beginning with **Y** or **y** on the standard input; a line beginning with **N** or **n** aborts the archive; with any other character **bar** reissues the prompt.

## SEE ALSO

**cpio**(1), **umask**(2), **bar**(5), **tar**(5), **dump**(8), **restore**(8)

## BUGS

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

There is no way to ask for the *n*th occurrence of a file.

The **u** option can be slow.

There is no way selectively to follow symbolic links.

When extracting tapes created with the **r** or **u** options, directory modification times may not be set correctly.

Files with names longer than 100 characters cannot be processed.

Filename substitution wildcards do not work for extracting files from the archive. To get around this, use a command of the form:

**bar xvf... /dev/rst0 'bar tf... /dev/rst0 | grep '*pattern*''**

If you specify '–' as the target file and the archive spans volumes, the request for a new volume may get lost.

NAME

basename, dirname – display portions of pathnames and filenames

SYNOPSIS

**basename** *string* [ *suffix* ]
**dirname** *string*

DESCRIPTION

**basename** deletes any prefix ending in / and the *suffix*, if present in *string*. It directs the result to the standard output, and is normally used inside substitution marks (` `) within shell procedures.

**dirname** delivers all but the last level of the path name in *string*.

EXAMPLES

This shell procedure invoked with the argument **/usr/src/bin/cat.c** compiles the named file and moves the output to **cat** in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

The following example will set the shell variable **NAME** to **/usr/src/cmd**:

```
NAME=‘dirname /usr/src/cmd/cat.c‘
```

SEE ALSO

**sh(1)**

## NAME

bc – arbitrary-precision arithmetic language

## SYNOPSIS

**bc** [ **–c** ] [ **–l** ] [ *filename*... ]

## DESCRIPTION

**bc** is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. **bc** takes input from any files given, then reads the standard input.

## OPTIONS

**–c** Compile only. **bc** is actually a preprocessor for **dc**(1), which it invokes automatically, unless the **–c** (compile only) option is present. In this case the **dc** input is sent to the standard output instead.

**–l** Is the name of an arbitrary precision math library.

## USAGE

### Comments

Enclosed in /* and */.

### Names

Simple variables: *l*, where, *l* is a lower-case letter.
Array elements: *l*[*expression*], where, *expression* is a legal **bc** expression.
The words **ibase**, **obase**, and **scale**.

### Other Operands

Arbitrarily long numbers with optional sign and decimal point.

| | |
|---|---|
| (*expression*) | |
| **sqrt** (*expression*) | |
| **length** (*expression*) | Number of significant decimal digits |
| **scale** (*expression*) | Number of digits right of decimal point |
| *l*(*expression*, | ..., *expression*) |

### Operators

| | |
|---|---|
| + – * / % ^ | (% is remainder; ^ is exponent) |
| ++ — | (prefix and postfix; apply to names) |
| == <= >= != < > | |
| = += –= *= /= %= ^= | |

### Statements

*expression*
{*statement* ; ... ; *statement*}
where, *statement* is a legal **bc** statement.
**if** (*expression*)*statement*
**while** ( **expression** ) **statement**
**for** ( *expression* ; *expression* ; *expression* ) *statement*
**null statement**
**break**
**quit**

### Function Definitions

**define** *l* ( *l* ,..., *l* ) {
**auto** *l* ,..., *l*
*statement* ;... *statement*
**return** ( *expression* ) }

### Functions in -l Math Library

| | |
|---|---|
| **s**(*x*) | sine |
| **c**(*x*) | cosine |
| **e**(*x*) | exponential |

| | |
|---|---|
| **l**(*x*) | log |
| **a**(*x*) | arctangent |
| **j**(*n*,*x*) | Bessel function |

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to **scale** influences the number of digits to be retained on arithmetic operations in the manner of **dc**(1). Assignments to **ibase** or **obase** set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. 'Auto' variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables empty square brackets must follow the array name.

## EXAMPLES

Define a function to compute an approximate value of the exponential function:

```
scale = 20
define
e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

Print approximate values of the exponential function of the first ten integers:

```
for(i=1; i<=10; i++) e(i)
```

## FILES

| | |
|---|---|
| **/usr/lib/lib.b** | mathematical library |
| **dc**(1) | desk calculator proper |

## SEE ALSO

**dc**(1)

*Games, Demos & Other Pursuits*

## BUGS

**for** statement must have all three *expression*'s.
**quit** is interpreted when read, not when executed.

NAME

biff – give notice of incoming mail messages

SYNOPSIS

**biff** [ y|n ]

DESCRIPTION

**biff** turns mail notification on or off for the terminal session. With no arguments, **biff** displays the current notification status for the terminal.

If notification is allowed, the terminal rings the bell and displays the header and the first few lines of each arriving mail message. **biff** operates asynchronously. For synchronized notices, use the **MAIL** variable of **sh**(1) or the **mail** variable of **csh**(1).

A '**biff y**' command can be included in your **.login** or **.profile** file for execution when you log in.

OPTIONS

**y** Allow mail notification for the terminal.

**n** Disable notification for the terminal.

FILES

**.login**
**.profile**

SEE ALSO

**cmdtool**(1), **csh**(1), **mail**(1), **sh**(1), **shelltool**(1), **sunview**(1), **comsat**(8C)

BUGS

You must have ownership the terminal to change its mail-notification status with **biff**, but windows running under **sunview**(1) are owned by the super-user. If you enable mail notification for the workstation console (in your **.login** or **.profile** file), incoming mail notices also appear on console windows running under **sunview**(1). See **shelltool**(1) or **cmdtool**(1) for details.

## NAME

binmail – an early program for processing mail messages

## SYNOPSIS

**/usr/bin/mail** [ **–ipq** ] [ **–f** *filename* ] *address*
**/usr/bin/mail** *recipient* ...

## DESCRIPTION

Note: This is the old version 7 UNIX system mail program. The default **mail** command, **/usr/ucb/mail** is described in **mail**(1).

**/usr/bin/mail** with no *address* prints a user's mail, message-by-message in last-in, first-out order. **/usr/bin/mail** accepts commands from the standard input to direct disposition messages.

When *address*es are named, **/usr/bin/mail** takes the standard input up to an EOF (or a line with just '.') and routes it through the mailer daemon to each *recipient*. See **sendmail**(8) for details. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *recipient* is a user name recognized by **login**(1), a network address or local mail alias, or a filename (see **aliases**(5) for details).

If there is any pending mail, **login** tells you there is mail when you log in. It is also possible to have the C shell, or the daemon **biff** tell you about mail that arrives while you are logged in.

To forward mail automatically, add the addresses of additional recipients to the **.forward** file in your home directory. Note: forwarding addresses must be valid, or the messages will bounce. (You cannot, for instance, reroute your mail to a new host by forwarding it to your new address if it is not yet listed in the YP aliases domain.)

## OPTIONS

**–i** Ignore interrupts.

**–p** Print messages without prompting for commands. Exit immediately upon receiving an interrupt.

**–q** Quit immediately upon interrupt.

**–f** *filename*
Use *filename* as if it were the mail file.

## USAGE

| | |
|---|---|
| **?** | Print a command summary. |
| **EOT(CTRL–D)** | Put unexamined mail back in the mail file and quit. |
| **!***command* | Escape to the shell to do *command*. |
| **–** | Go back to previous message. |
| **+** | Go on to next message. |
| **NEWLINE** | Go on to next message. |
| **d** | Delete message and go on to the next. |
| **dq** | Delete message and quit. |
| **m** [ *person* ]... | Mail the message to the named *recipients* (yourself is default). |
| **n** | Go on to next message. |
| **p** | Print message (again). |
| **q** | Same as EOT . |

**s** [*filename*]... Save the message in the named *files* ('**mbox**' default). If saved successfully, remove it from the list and go on to the next message.

**w** [*filename* ]... Save the message, without a header, in the named *files* ('**mbox**' default). If saved successfully, remove it from the list and go on to the next message.

**x** Exit without changing the mail file.

FILES

| | |
|---|---|
| **/etc/passwd** | to identify sender and locate address |
| **/var/spool/mail/*** | incoming mail for user * |
| **/usr/ucb/mail** | routes input through daemon to *recipient*s |
| **mbox'** | saved mail |
| **/tmp/ma*** | temp file |
| **/var/spool/mail/*.lock** | lock for mail directory |
| **dead.letter** | unmailable text is saved here |
| **$HOME/.forward** | list of forwarding recipients |

SEE ALSO

**biff**(1), **csh**(1), **des**(1), **login**(1), **mail**(1), **uucp**(1C), **uux**(1C), **write**(1), **xsend**(1), **crypt**(3), **aliases**(5), **sendmail**(8)

BUGS

Race conditions sometimes result in a failure to remove a lock file.

The super-user can read your mail, unless it is encrypted by **des**(1), **xsend**(1), or **crypt**(3). Even if you encrypt it, the super-user can delete it.

**NAME**

cal – display a calendar

**SYNOPSIS**

**cal** [ [ *month* ] *year* ]

**DESCRIPTION**

**cal** displays a calendar for the specified year. If a month is also specified, a calendar for that month only is displayed. If neither is specified, a calendar for the present month is printed.

*year* can be between 1 and 9999. Be aware that '**cal 78**' refers to the early Christian era, not the 20th century. Also, the year is always considered to start in January, even though this is historically naive.

*month* is a number between 1 and 12.

The calendar produced is that for England and her colonies.

Try September 1752.

## NAME

calendar – a simple reminder service

## SYNOPSIS

**calendar** [ – ]

## DESCRIPTION

**calendar** consults the file **calendar** in the current directory and displays lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates — such as 'Dec. 7,' 'december 7,' and '12/7' — are recognized, but '7 December' or '7/12' are not. If you give the month as '*' with a date — for example, ''* 1'' — that day in any month will do. On weekends "tomorrow" extends through Monday.

When the optional '–' argument is present, **calendar** does its job for every user who has a file **calendar** in his login directory and sends him any positive results by **mail**(1). Normally this is done daily in the wee hours under control of **cron**(8).

The file **calendar** is first run through the C preprocessor, **/lib/cpp**, to include any other calendar files specified with the usual **#include** syntax. Included calendars are usually shared by all users, and maintained by the system administrator.

## FILES

**/calendar**
**/usr/lib/calendar** to figure out today's and tomorrow's dates
**/etc/passwd**
**/tmp/cal***
**/lib/cpp**
**/usr/bin/egrep**
**/usr/bin/sed**
**/usr/bin/mail**

## SEE ALSO

**at**(1), **mail**(1), **cron**(8)

## BUGS

**calendar** 's extended idea of "tomorrow" does not account for holidays.

Problems may occur when there is no **/etc/passwd** file on the local host.

## NAME

cat – concatenate and display

## SYNOPSIS

**cat** [ – ] [ **–benstuv** ] [ *filename*... ]

## SYSTEM V SYNOPSIS

**cat** [ – ] [ **–estuv** ] [ *filename*... ]

## DESCRIPTION

**cat** reads each *filename* in sequence and displays it on the standard output. Thus:

**example% cat goodies**

displays the contents of **goodies** on the standard output, and

**example% cat** *filename1 filename2* **>** *filename3*

concatenates the first two files and places the result on the third.

If no filename argument is given, or if the argument '–' is given, **cat** reads from the standard input. If the standard input is a terminal, input is terminated by an EOF signal, usually CTRL-D.

## OPTIONS

**–b** Number the lines, as **–n**, but omit the line numbers from blank lines.

**–e** Display non-printing characters, as **–v**, and in addition display a **$** character at the end of each line.

**–n** Precede each line output with its line number.

**–s** Substitute a single blank line for multiple adjacent blank lines.

**–t** Display non-printing characters, as **–v**, and in addition display TAB characters as **^I** (CTRL-I).

**–u** Unbuffered. If **–u** is not used, output is buffered in blocks, or line-buffered if standard output is a terminal.

**–v** Display non-printing characters (with the exception of TAB and NEWLINE characters) so that they are visible. Control characters print like **^X** for CTRL-X; the DEL character (octal 0177) print as **'^?'**. Non-ASCII characters (with the high bit set) are displayed as **M–***x* where **M–** stands for **'meta'** and *x* is the character specified by the seven low order bits.

## SYSTEM V OPTIONS

**–e** If the **–v** option is specified, display a **$** character at the end of each line.

**–s** Suppress messages about files which cannot be opened.

**–t** If the **–v** option is specified, displays TAB characters as **^I** (CTRL-I) and FORMFEED characters as **^L** (CTRL-I).

**–v** Display non-printing character (with the exception of TAB, NEWLINE, and FORMFEED characters) so that they are visible.

## SEE ALSO

**cp**(1), **ex**(1), **more**(1), **pg**(1V), **pr**(1V), **tail**(1)

## BUGS

Beware of **'cat a b >a'** and **'cat a b >b'**, which destroy the input files before reading them.

NAME

cb – a simple C program beautifier

SYNOPSIS

**cb** [ **–s** ] [ **–j** ] [ **–l** *leng* ] [ *filename*... ]

DESCRIPTION

**cb** reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code.

**indent**(1) is preferred.

OPTIONS

With no options, **cb** preserves all user NEWLINES.

**–s** Standard C style. Canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*.

**–j** Split lines are put back together.

**–l** *leng* Split lines longer than *leng*.

SEE ALSO

**indent**(1)

B.W. Kernighan and D.M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978

BUGS

Punctuation hidden in preprocessor statements can cause indentation errors.

## NAME

cc – C compiler

## SYNOPSIS

**cc** [ **–a** ] [ **–align** *_block* ] [ **–B** *binding* ] [ **–c** ] [ **–C** ] [ **–dryrun** ] [ **–D***name* [**=***def* ] ] [ **–E** ]
[ *float_option* ] [ **–fsingle** ] [ **–g** ] [ **–go** ] [ **–help** ] [ **–I***pathname* ] [ **–J** ] [ **–l***library* ]
[ **–L***directory* ] [ **–M** ] [ **–misalign** ] [ **–o** *outputfile* ] [ **–O**[*level*] ] [ **–p** ] [ **–P** ] [ **–pg** ] [ **–pic** ]
[ **–PIC** ] [ **–pipe** ] [ **–Qoption** *prog opt* ] [ **–Qpath** *pathname* ] [ **–Qproduce** *sourcetype* ] [ **–R** ]
[ **–S** ] [ *target_arch* ] [ **–temp=***directory* ] [ **–time** ] [ **–U***name* ] [ **–v** ] [ **–w** ] *sourcefile* ...

## SYSTEM V SYNOPSIS

**/usr/5bin/cc** *arguments*

Note: *arguments* to **/usr/5bin/cc** are identical to those listed above.

## DESCRIPTION

**cc** is the C compiler. It translates programs written in the C programming language into executable load modules, or into relocatable binary programs for subsequent loading with the **ld**(1) link editor.

In addition to the many options, **cc** accepts several types of filename arguments. For instance, files with names ending in **.c** are taken to be C source programs. They are compiled, and each resulting object program is placed in the current directory. The object file is named after its source file — the suffix **.o** replacing **.c** in the name of the object. In the same way, files whose names end with **.s** are taken to be assembly source programs. They are assembled, and produce **.o** files. Filenames ending in **.il** are taken to be inline expansion code template files; these are used to expand calls to selected routines in-line when code optimization is enabled. See FILES, below for a complete list of compiler-related filename suffixes.

Other arguments refer to assembler or loader options, object programs, or object libraries. Unless **–c**, **–S**, **–E** **–P** or **–Qproduce** is specified, these programs and libraries, together with the results of any specified compilations or assemblies, are loaded (in the order given) to produce an output file named **a.out**. You can specify a name for the executable by using the **–o** option.

If a single C program is compiled and loaded all at once, the intermediate file is deleted.

## OPTIONS

When debugging or profiling objects are compiled using the **–g** or **–pg** options, respectively, the **ld** command for linking them should also contain the appropriate option.

See **ld**(1) for link-time options.

**–a** Insert code to count how many times each basic block is executed. Invokes a run-time recording mechanism that creates a **.d** file for every **.c** file (at normal termination). The **.d** file accumulates execution data for the corresponding source file. The **tcov**(1) utility can then be run on the source file to generate statistics about the program. Since this option entails some optimization, it is incompatible with **–g**.

**–align** *_block* Force the global uninitialized data symbol *block* to be page-aligned by increasing its size to a whole number of pages, and placing its first byte at the beginning of a page.

**–B** *binding* Specify whether bindings of libraries for linking are **static** or **dynamic**, indicating whether libraries are non-shared or shared, respectively.

**–c** Suppress linking with **ld**(1) and produce a **.o** file for each source file. A single object file can be named explicitly using the **–o** option.

**–C** Prevent the C preprocessor, **cpp**(1), from removing comments.

**–dryrun** Show but do not execute the commands constructed by the compilation driver.

**–D***name*[**=***def*] Define a symbol *name* to the C preprocessor (**cpp**(1)). Equivalent to a **#define** directive in the source. If no *def* is given, *name* is defined as '**1**'.

**–E** Run the source file through **cpp**(1), the C preprocessor, only. Sends the output to the standard output, or to a file named with the **–o** option. Includes the **cpp** line numbering

information. (See also, the **–P** option.)

*float_option* Floating-point code generation option. Can be one of:

**–f68881** Generate in-line code for Motorola MC68881 floating-point processor (supported only on Sun-3 systems).

**–ffpa** Generate in-line code for Sun Floating Point Accelerator (supported only on Sun-3 systems).

**–fsky** Generate in-line code for Sky floating-point processor (supported only on Sun-2 systems).

**–fsoft** Generate software floating-point calls. Supported only on Sun-2 and Sun-3 systems, for which it is the default.

**–fswitch** Run-time-switched floating-point calls. The compiled object code is linked at runtime to routines that support one of the above types of floating point code. This was the default in previous releases. Only for use with programs that are floating-point intensive, and must be portable to machines with various floating-point hardware options (supported only on Sun-2 and Sun-3 systems).

**–fsingle** (Sun-2, Sun-3 and Sun-4 systems)
Use single-precision arithmetic in computations involving only **float** expressions. Do not convert everything to **double**, which is the default. Note: floating-point *parameters* are still converted to double precision, and *functions* returning values still return double-precision values.

Although not standard C, certain programs run much faster using this option. Be aware that some significance can be lost due to lower-precision intermediate values.

**–g** Produce additional symbol table information for **dbx**(1) and **dbxtool**(1) and pass the **–lg** flag to **ld**(1). When this option is given, the **–O** and **–R** options are suppressed.

**–go** Produce additional symbol table information for **adb**(1). When this option is given, the **–O** and **–R** options are suppressed.

**–help** Display helpful information about **cc**.

**–I***pathname* Add *pathname* to the list of directories in which to search for **#include** files with relative filenames (not beginning with slash /). The preprocessor first searches for **#include** files in the directory containing *sourcefile*, then in directories named with **–I** options (if any), and finally, in **/usr/include**.

**–J** Generate 32-bit offsets in **switch** statement labels (supported only on Sun-2 and Sun-3 systems).

**–l***library* Link with object library *library* (for **ld**(1)).

**–L***directory* Add *directory* to the list of directories containing object-library routines (for linking using **ld**(1).

**–M** Run only the macro preprocessor on the named C programs, requesting that it generate makefile dependencies and send the result to the standard output (see **make**(1) for details about makefiles and dependencies).

**–misalign** Generate code to allow loading and storage of misaligned data (Sun-4 systems only).

**–o** *outputfile* Name the output file *outputfile*. *outputfile* must have the appropriate suffix for the type of file to be produced by the compilation (see FILES, below). *outputfile* cannot be the same as *sourcefile* (the compiler will not overwrite the source file).

**–O**[*level*] Optimize the object code. Ignored when either **–g**, **–go**, or **–a** is used. On Sun-2 and Sun-3 systems, **–O** with the *level* omitted is equivalent to **–O1**; on Sun-4 systems, it is equivalent to **–O2**. on Sun386i systems, all levels are the same as 1. *level* is one of:

- **1** Do postpass assembly-level optimization only.
- **2** Do global optimization prior to code generation, including loop optimizations, common subexpression elimination, copy propagation, and automatic register allocation. **–O2** does not optimize references to or definitions of external or indirect variables.
- **3** Same as **–O2**, but optimize uses and definitions of external variables. **–O3** does not trace the effects of pointer assignments. Neither **–O3** nor **–O4** should be used when compiling either device drivers, or programs that modify external variables from within signal handlers.
- **4** Same as **–O3**, but trace the effects of pointer assignments.

**–p** Prepare the object code to collect data for profiling with **prof**(1). Invokes a run-time recording mechanism that produces a **mon.out** file (at normal termination).

**–P** Run the source file through **cpp**(1), the C preprocessor, only. Puts the output in a file with a **.i** suffix. Does not include **cpp**-type line number information in the output.

**–pg** Prepare the object code to collect data for profiling with **gprof**(1). Invokes a run-time recording mechanism that produces a **gmon.out** file (at normal termination).

**–pic** Produce position-independent code. Each reference to a global datum is generated as a dereference of a pointer in the global offset table. Each function call is generated in pc-relative addressing mode through a procedure linkage table. The size of the global offset table is limited to 64K on MC68000-family processors, or to 8K on SPARC processors.

**–PIC** Like **–pic**, but allows the global offset table to span the range of 32-bit addresses in those rare cases where there are too many global data objects for **–pic**.

**–pipe** Use pipes, rather than intermediate files, between compilation stages. (Very cpu-intensive.)

**–Qoption** *prog opt*
Pass the option *opt* to the program *prog*. The option must be appropriate to that program and may begin with a minus sign. *prog* can be one of: **as, cpp, inline,** or **ld.**

**–Qpath** *pathname*
Insert directory *pathname* into the compilation search path (to use alternate versions of programs invoked during compilation). This path will also be searched first for certain relocatable object files that are implicitly referenced by the compiler driver (such files as ***crt*.o** and **bb_link.o** ).

**–Qproduce** *sourcetype*
Produce source code of the type *sourcetype*. *sourcetype* can be one of:

- **.c** C source (from **bb_count**).
- **.i** Preprocessed C source from **cpp**(1).
- **.o** Object file from **as**(1).
- **.s** Assembler source (from **ccom, inline**(1) or **c2**).

**–R** Merge data segment with text segment for **as**(1). Data initialized in the object file produced by this compilation is read-only, and (unless linked with **ld –N**) is shared between processes. Ignored when either **–g** or **–go** is used.

**–S** Do not assemble the program but produce an assembly source file.

*target_arch* Compile object files for the specified processor architecture. Unless used in conjunction with one of the Sun Cross-Compilers, correct programs can be generated only for the architecture of the host on which the compilation is performed. *target_arch* can be one of:

**–sun2** Produce object files for a Sun-2 system.
**–sun3** Produce object files for a Sun-3 system.
**–sun4** Produce object files for a Sun-4 system.

**–temp=***directory*
Set directory for temporary files to be *directory*.

**–time** Report execution times for the various compilation passes.

**–U***name* Remove any initial definition of the **cpp**(1) symbol *name*. (Inverse of the **–D** option.)

**–v** Verbose. Print the version number of the compiler and the name of each program it executes.

**–w** Do not print warnings.

ENVIRONMENT

**FLOAT_OPTION** (Sun-2, Sun-3, Sun-4 systems only.) When no floating-point option is specified, the compiler uses the value of this environment variable (if set). Recognized values are: **f68881, ffpa, fsky, fswitch** and **fsoft.**

FILES

| | |
|---|---|
| **a.out** | executable output file |
| *file*.**a** | library of object files |
| *file*.**c** | C source file |
| *file*.**d** | **tcov**(1) test coverage input file (Sun-2, Sun-3, Sun-4 systems only) |
| *file*.**i** | C source file after preprocessing with **cpp**(1) |
| *file*.**il** | *inline* expansion file |
| *file*.**o** | object file |
| *file*.**s** | assembler source file |
| *file*.**S** | assembler source for **cpp**(1) |
| *file*.**tcov** | output from **tcov**(1) (Sun-2, Sun-3, Sun-4 systems only) |
| **/usr/lib/c2** | object code optimizer |
| **/usr/lib/ccom** | compiler |
| **/usr/lib/compile** | compiler command-line processing driver |
| **/usr/lib/cpp** | macro preprocessor |
| **/usr/lib/crt0.o** | runtime startup code |
| **/usr/lib/Fcrt1.o** | startup code for **–fsoft** option (Sun-2, Sun-3, Sun-4 systems only) |
| **/usr/lib/gcrt0.o** | startup for profiling with **gprof**(1) |
| **/usr/lib/libc.a** | standard library, see **intro**(3) |
| **/usr/lib/mcrt0.o** | startup for profiling with **prof**(1) **intro**(3) |
| **/usr/lib/Mcrt1.o** | startup code for **–f68881** option (for Sun-3 systems) |
| **/usr/lib/Scrt1.o** | startup code for **–fsky** option (for Sun-2 systems) |
| **/usr/lib/Wcrt1.o** | startup code for **–ffpa** option (for Sun-3 systems) |
| **/usr/include** | standard directory for **#include** files |
| **/usr/lib/bb_link.o** | basic block counting routine |
| **/usr/lib/cg** | code generator used with **/usr/lib/iropt** |
| **/usr/lib/libc_p.a** | profiling library, see **gprof**(1) or **prof**(1) |
| **/usr/lib/inline** | inline expander of library calls |
| **/usr/lib/iropt** | intermediate representation optimizer |
| **/usr/lib/libm.a** | math library |
| **/usr/5lib/libc.a** | System V standard compatibility library, see **intro**(3V) |
| **/usr/5lib/libc_p.a** | System V profiling library, see **gprof**(1) or **prof**(1) |

| | |
|---|---|
| **/tmp/*** | compiler temporary files |
| **mon.out** | file produced for analysis by **prof**(1) |
| **gmon.out** | file produced for analysis by **gprof**(1) |

SEE ALSO

**adb**(1), **ar**(1V), **as**(1), **cpp**(1), **dbx**(1), **dbxtool**(1), **gprof**(1), **inline**(1), **ld**(1), **lint**(1V), **make**(1), **prof**(1), **tcov**(1), **intro**(3), **intro**(3V), **monitor**(3)

*Floating Point Programmers Guide*
*Programming Utilities and Libraries*
B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978

DIAGNOSTICS

The diagnostics produced by the C compiler are intended to be self-explanatory. Occasional obscure messages may be produced by the preprocessor, assembler, or loader.

BUGS

The program context given in syntax error messages is taken from the input text *after* the C preprocessor has performed substitutions. Therefore, error messages involving syntax errors in or near macro references or manifest constants may be misleading.

Compiling with optimization level 2 or greater may produce incorrect object code if tail-recursion elimination is applied to functions called with fewer actual parameters (arguments) than the number of formal parameters in the function's definition. Such parameter-count mismatches can be detected using **lint**(1V).

NAME

cd – change working directory

SYNOPSIS

**cd** [ *directory* ]

DESCRIPTION

*directory* becomes the new working directory. The process must have execute (search) permission in *directory*. If **cd** is used without arguments, it returns you to your login directory. In **csh**(1) you may specify a list of directories in which *directory* is to be sought as a subdirectory if it is not a subdirectory of the current directory; see the description of the **cdpath** variable in **csh**(1).

SEE ALSO

**csh**(1), **pwd**(1), **sh**(1)

## NAME

cdc – change the delta commentary of an SCCS delta

## SYNOPSIS

**/usr/sccs/cdc** –**r***SID* [ –**m** [ *mrlist* ] ] [ –**y** [ *comment* ] ] *filename* ...

## DESCRIPTION

**cdc** changes the delta commentary, for the SID (SCCS ID) specified by the –**r** option, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified by the **delta**(1) command.

If a directory is named, **cdc** behaves as though each file in the directory were specified, except that non-SCCS files (last component of the path name does not begin with 's.') and unreadable files are silently ignored. If a *filename* of '-' is given, each line of the standard input is taken to be the name of an SCCS file to be processed, and the –**m** and –**y** options must be used.

### Permissions

If you made the delta, you can normally change its delta commentary, or if you own the file and directory, and have write permission, you can modify the delta commentary.

## OPTIONS

Arguments to **cdc**, which may appear in any order, consist of options and file names.

Each option applies independently to each *filename*.

–**r***SID* Specify the (SID) string of a delta for which the delta commentary is to be changed.

–**m** [ *mrlist* ] If the SCCS file has the **v** flag set (see **admin**(1)), a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the –**r** option may be supplied (see EXAMPLES). A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of **delta**. In order to delete an MR, precede the MR number with the character '!'. If the MR to be deleted is currently in the list of MRs, it is removed and changed into a comment line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If –**m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see the –**y** option).

MRs in a list are separated by SPACE and/or TAB characters. An unescaped NEWLINE character terminates the MR list.

Note: if the **v** flag has a value it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, **cdc** terminates and the delta commentary remains unchanged.

–**y** [ *comment* ] Arbitrary text used to replace a *comment* already existing for the specified delta. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If –**y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped NEWLINE character terminates the *comment* text.

EXAMPLES

The following command:

**example% cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file**

adds **bl78-12345** and **bl79-00001** to the MR list, removes **bl77-54321** from the MR list, and adds the comment **trouble** to **delta 1.6** of **s.file**.

The command:

**example% cdc -r1.6 s.file**
**MRs? !bl77-54321 bl78-12345 bl79-00001**
**comments? trouble**

does the same thing.

FILES

**x-file** (see **delta**(1))
**z-file** (see **delta**(1))

SEE ALSO

**admin**(1), **comb**(1), **delta(1)**, **get**(1), **help**(1), **prs(1)**, **sccs**(1), **sccsdiff**(1), **val**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries.*

DIAGNOSTICS

Use **help**(1) for explanations.

## NAME

cflow– generate a flow graph for a C program

## SYNOPSIS

**cflow** [**–r**] [**–ix**] [ **–i_** ] [ **–d***num* ] *filenames*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**cflow** analyzes a collection of C, **yacc**, **lex**, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in **.y**, **.l**, **.c**, and **.i** are **yacc**'d, **lex**'d, and C-preprocessed output files, respectively (bypassed for **.i** files) as appropriate and then run through the first pass of **lint**(1V). (The **–I**, **–D**, and **–U** options of the C-preprocessor are also understood.) Files suffixed with **.s** are assembled and information is extracted (as in **.o** files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (that is, line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the **–i** inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (for example, **char ***), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (for example, *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only **< >** is printed.

## SYSTEM V DESCRIPTION

The System V version of **cflow** in **/usr/5bin/cflow** makes the C preprocessor, **cpp**(1) search in **/usr/5include** for include files before it searches in **/usr/include**.

## OPTIONS

The following options are interpreted by **cflow :**

**–r** Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

**–ix** Include external and static data symbols. The default is to include only functions in the flowgraph.

**–i_** Include names that begin with an underscore. The default is to exclude these functions (and data if **–ix** is used).

**–d***num* The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

## EXAMPLE

As an example, given the following in **file.c**:

```
int i;
main()
{
        f();
        g();
        f();
}
```

```
f()
{
        i = h();
}
```

the command:

**cflow –ix file.c**

produces the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the –e option of **pr**(1V) can be used to compress the tab expansion to something less than every eight spaces.

FILES

**/usr/5bin/cflow**
**/usr/include**

SEE ALSO

**as**(1), **cc**(1V), **cpp**(1), **lex**(1), **lint**(1V), **nm**(1), **pr**(1V), **yacc**(1)

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used, such as, the C-preprocessor.

BUGS

Files produced by **lex** and **yacc** cause the reordering of line number declarations which can confuse **cflow** . To get proper results, feed **cflow** the **yacc** or **lex** input.

## NAME

checknr – check nroff and troff input files; report possible errors

## SYNOPSIS

**checknr** [ **–fs** ] [ **–a** *.x1 .y1 .x2 .y2 ... .xn .yn* ] [ **–c** *.x1 .x2 .x3 ... .xn* ] [ *filename ...* ]

## AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**checknr** checks a list of **nroff**(1) or **troff**(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, **checknr** checks the standard input. Delimiters checked are:

- Font changes using **\f***x* ... **\fP**.
- Size changes using **\s***x* ... **\s0**.
- Macros that come in open ... close forms, for example, the **.TS** and **.TE** macros which must always come in pairs.

**checknr** knows about the **ms**(7) and **me**(7) macro packages.

**checknr** is intended to be used on documents that are prepared with **checknr** in mind. It expects a certain document writing style for **\f** and **\s** commands, in that each **\f***x* must be terminated with **\fP** and each **\s***x* must be terminated with **\s0**. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from **checknr**. Since it is probably better to use the **\fP** and **\s0** forms anyway, you should think of this as a contribution to your document preparation style.

## OPTIONS

**–f** Ignore **\f** font changes.

**–s** Ignore **\s** size changes.

**–a** *.x1 .y1...*

Add pairs of macros to the list. The pairs of macros are assumed to be those (such as **.DS** and **.DE**) that should be checked for balance. The **–a** option must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair **.BS** and **.ES**, use '**–a.BS.ES**'

**–c** *.x1 ...*

Define commands which **checknr** would otherwise complain about as undefined.

## SEE ALSO

**eqn**(1), **nroff**(1), **troff**(1), **me**(7), **ms**(7)

## BUGS

There is no way to define a one-character macro name using the **–a** option.

NAME

chgrp – change the group ownership of a file

SYNOPSIS

**chgrp** [ **–f** ] [ **–R** ] *group-filename*...

DESCRIPTION

**chgrp** changes the group-ID of the *filename*s given as arguments to *group*. The group may be either a decimal GID or a group name found in the group- ID file, **/etc/group**.

You must belong to the specified group and be the owner of the file, or be the super-user.

OPTIONS

**–f** Force. Do not report errors.

**–R** Recursive. **chgrp** descends through the directory, and any subdirectories, setting the specified group-ID as it proceeds. When symbolic links are encountered, their group is changed, but they are not traversed.

FILES

**/etc/group**

SEE ALSO

**chown**(2), **group**(5), **passwd**(5)

**NAME**

chkey – change your encryption key

**SYNOPSIS**

**chkey**

**DESCRIPTION**

**chkey** prompts the user for their login password, and uses it to encrypt a new encryption key for the user to be stored in the **publickey**(5) database.

**SEE ALSO**

**keylogin**(1), **publickey**(5), **keyserv**(8C), **newkey**(8)

## NAME

chmod – change the permissions mode of a file

## SYNOPSIS

**chmod** [ **–fR** ] *mode filename* ...

## DESCRIPTION

Change the permissions (mode) of a file or files. Only the owner of a file (or the super-user) may change its mode.

The mode of each named file is changed according to *mode*. which may be absolute or symbolic, as follows.

### Absolute Modes

An absolute *mode* is an octal number constructed from the OR of the following modes:

**400** Read by owner.
**200** Write by owner.
**100** Execute (search in directory) by owner.
**040** Read by group.
**020** Write by group.
**010** Execute (search) by group.
**004** Read by others.
**002** Write by others.
**001** Execute (search) by others.
**4000** Set user ID on execution.
**2000** Set group ID on execution (this bit is ignored if the file is a directory; it may be set or cleared only using symbolic mode).
**1000** Sticky bit, (see **chmod** (2) for more information).

### Symbolic Modes

A symbolic *mode* has the form:

[ *who* ] *op permission* [ *op permission* ] ...

*who* is a combination of:

**u** User's permissions.
**g** Group permissions.
**o** Others.
**a** All, or **ugo**.

If *who* is omitted, the default is **a**. but the setting of the file creation mask (see **umask** in **sh**(1) or **csh**(1) for more information) is taken into account. When *who* is omitted, **chmod** will not override the restrictions of your user mask.

*op* is one of:

**+** To add the *permission*.
**–** To remove the *permission*.
**=** To assign the permission explicitly (all other bits for that category, owner, group, or others, will be reset).

*permission* is any combination of:

**r** Read.
**w** Write.
**x** Execute.
**X** Give execute permission if the file is a directory or if there is execute permission for one of the other user classes.

**s** Set owner- or group-ID. This is only useful with **u** or **g**. Also, the set group-ID bit of a directory may only be modified with + or −.

**t** Set the sticky bit to save program text between processes.

The letters **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode for the user-class.

Omitting *permission* is only useful with '=', to take away all permissions.

Multiple symbolic modes, separated by commas, may be given. Operations are performed in the order specified.

SYSTEM V DESCRIPTION

If *who* is omitted in a symbolic mode, it does not take the file creation mask into account, but acts as if *who* were **a**.

OPTIONS

**−f** Force. **chmod** will not complain if it fails to change the mode of a file.

**−R** Recursively descend through directory arguments, setting the mode for each file as described above. When symbolic links are encountered, their mode is not changed and they are not traversed.

EXAMPLES

The first example denies write permission to others, the second makes a file executable by all if it is executable by anyone:

**chmod o−w file**
**chmod +X file**

SEE ALSO

**csh**(1), **ls**(1V), **sh**(1), **chmod**(2), **chown**(8)

NAME

clear – clear the terminal screen

SYNOPSIS

**clear**

DESCRIPTION

**clear** clears your screen if this is possible. It looks in the environment for the terminal type and then in **/etc/termcap** to figure out how to clear the screen.

FILES

**/etc/termcap** terminal capability data base

NAME

clear_colormap – clear the colormap to make console text visible

SYNOPSIS

**clear_colormap** [ **–no** ] [ **–f** *framebuffer* ]

DESCRIPTION

**clear_colormap** ensures that text displayed on the console is visible. If no options are specified it clears the frame buffer and initializes the first two colormap entries. If the frame buffer has an overlay plane it is also cleared, its colormap is initialized, and the overlay enable plane is set so that the entire overlay plane is displayed.

OPTIONS

**–n** Do not clear the frame buffer or overlay plane.

**–o** Do not clear the overlay plane, initialize its colormap, or modify the overlay enable plane.

**–f** *framebuffer*

Operate on frame buffer device *framebuffer* instead of the default, **/dev/fb**.

FILES

**/dev/fb**

NAME

clear_functions – reset the selection service to clear stuck function keys

SYNOPSIS

**clear_functions**

DESCRIPTION

**clear_functions** instructs the selection service that no function keys are currently depressed. It is useful in cases where erroneous programs have reported a key press but not the corresponding release. The usual symptom for this situation is that all selections are secondary (underscored rather than inverted), even though no function keys are down.

FILES

**/usr/bin/selection_svc**

SEE ALSO

*SunView 1 Beginner's Guide*

NAME

click – enable or disable the keyboard's keystroke click

SYNOPSIS

**click** [ **–y** ] [ **–n** ] [ **–d** *keyboard-device* ]

DESCRIPTION

Change the setting of the audible keyboard click. The default is no keyboard click. If you want to turn clicking on then a good place to do it is in **/etc/rc.local**.

Only keyboards that support a clicker respond to this command. At the time of this writing, the only keyboards known to have a clicker are the Sun-3 and Sun386i systems keyboards.

OPTIONS

**–y** Yes, enable clicking.

**–n** No, disable clicking.

**–d** *keyboard-device*
Specify the keyboard device being set. The default is **/dev/kbd**.

FILES

**/etc/rc.local**
**/dev/kbd**

SEE ALSO

**kbd**(4S)

DIAGNOSTICS

A short help message is printed if an unknown flag is specified or if the **–d** switch is used and the keyboard device name is not supplied. A diagnostic is printed if the keyboard device name can't be opened or is not a keyboard type device.

BUGS

There is no way to determine the state of the keyboard click setting.

NAME

clock – display the time in an icon or window

SYNOPSIS

**clock** [ **–s** ] [ **–t** ] [ **–r** ] [ **–d mdyaw** ] [ **–f** ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**clock** is a SunView utility that displays the current time in a window. When open, **clock** shows the date and time in text format. When closed, **clock** displays a clock face.

Note: In previous releases **clock** was known as **clocktool**. In the current release, **clocktool** is a symbolic link to **clock**.

OPTIONS

**–r** causes **clock** to use a square face with roman numerals in the iconic state. This replaces the default round clock face.

**–d** display date information in a small area just below the clock face. The date information to be displayed may include:

- **m** the month,
- **d** the day of the month (1-31),
- **y** the year,
- **a** the string AM or PM, as appropriate,
- **w** the day of the week (Sun–Sat).

There is only room for 3 of these, but any 3 may be displayed in any sequence.

**–f** Display the date and day of week on the clock face.

**–s** start **clock** with the seconds turned on. By default, the clock starts with seconds turned off, and updates every minute. With seconds turned on, it updates every second, and, if iconic, displays a second hand.

**–t** Test mode — ignore the real time, and instead run in a loop continuously incrementing the time by one minute and displaying it.

**clock** also accepts all of the generic tool arguments discussed in **sunview**(1).

USAGE

**clock** listens for keyboard input while open. It recognizes the following characters:

**s** or **S** Enable or disable the display of seconds.

**t** or **T** Enable or disable "test" mode.

FILES

**/usr/lib/fonts/fixedwidthfonts/sail.r.6**
font for day-of-month clock-face display

SEE ALSO

**sunview**(1), **date**(1V)

BUGS

If you reset the system time, **clock** will not reflect the new time until you change its state from open to icon, or vice versa. To reset the system time, see **date**(1V).

The date display does not go well with the round clock face.

NAME

cluster – find the Application SunOS or Developer's Toolkit optional cluster containing a file

SYNOPSIS

**cluster** [ *filename* ]

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**cluster** finds the optional software cluster that contains a specified file. If the specified file is contained in one of the clusters in Application SunOS or Developer's Toolkit, the name of the cluster will be printed on standard output.

Without arguments, **cluster** displays a summary of the clusters in Application SunOS and Developers Toolkit, including the load state and size of each cluster.

EXAMPLES

To find the name of the cluster that contains the **spell** command:

```
example% cluster spell
spellcheck
example%
```

To display a summary of the clusters in Application SunOS and Developer's Toolkit:

```
% cluster
Application SunOS Clusters:
   available              cluster   size (bytes)
   ---------  -------     ----
   yes        accounting  265K
   no         advanced_admin 501K
   ...

Developer's Toolkit Clusters:
  availablecluster        size (bytes)
  ---------  -------      ----
   no        base_devel   6907K
   ...

space used by clusters: 6021K bytes
total space remaining: 20432K bytes
```

A cluster is available if it has been "loaded" using **load**(1) or if it has been "mounted" across the network.

FILES

**/usr/lib/load/*** data files

SEE ALSO

**load**(1), **unload**(1), **toc**(5)

*Sun386i System Setup and Maintenance*

DIAGNOSTICS

**The file** *filename* **is not in any of the optional software clusters.**

The specified file is not part of the Application SunOS or Developer's Toolkit.

## NAME

cmdtool – run a shell (or program) using the SunView text facility

## SYNOPSIS

**cmdtool** [ **–C** ] [ **–M** *bytes* ] [ **–P** *count* ] [ *generic-tool-arguments* ] [ *program* [ *program-arguments* ] ]

## AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**cmdtool** is the standard *SunView* support facility for shells and other programs. When invoked, **cmdtool** runs a program (usually a shell) in a text-based command subwindow. Characters typed on the keyboard are inserted at the caret. If the program is a shell, that shell accepts and runs commands in the usual manner. **cmdtool** also supports programs that perform cursor motions directly, such as **vi**(1).

The text of the current command line can be edited using normal **Text Edit** functions. The command subwindow displays a log of the session, which can be scrolled through using the scrollbar (unless the command does cursor motion). This log can be edited, and saved by choosing the '**Store as New File**' item in the text facility's pop-up menu. The '**Split View**' command, also in the pop-up menu, can be used to create two or more independent views of the log.

## OPTIONS

**–C** Console **cmdtool**. Display console messages in this **cmdtool**, which might otherwise appear in unexpected places on the workstation screen. Since a **cmdtool** window can be scrolled, console error messages can be recorded for later examination.

**–M** *bytes*
Set the log to wrap-around after the indicated number of *bytes*.

**–P** *count*
Checkpoint the log after every set of *count* editing operations.

*generic-tool-arguments*
**cmdtool** accepts the generic tool arguments listed in **sunview**(1).

*program* [ *program-arguments* ]
If a *program* argument is present, **cmdtool** runs it and passes any remaining arguments to that *program*. If no *program* is given, **cmdtool** runs the program indicated by the **SHELL** environment variable, or **/usr/bin/sh** by default.

## USAGE

Refer to *SunView 1 Beginner's Guide* for details on how to use **cmdtool**.

### Defaults Options

The following options can be configured as default settings using **defaultsedit**(1).

**/Tty/Append_only_log**
When set to **TRUE** (the standard default) only command lines can be edited. **FALSE** allows the entire log to be edited. (See also the description of **Enable Edit** below.)

**/Tty/Insert_makes_caret_visible**
This entry allows you to specify the method for displaying the editing caret.

| | |
|---|---|
| **Same_as_for_text** | Use the setting specified in the defaults for the **Text** category (the standard default). |
| **If_auto_scroll** | If the caret is showing, and an inserted NEWLINE would position it below the bottom of the screen (as determined by **/Text/Lower_context**), the text is scrolled to keep the caret showing. The number of lines scrolled is determined by the **/Text/Auto_scroll_by** default. (See **textedit**(1) for more information.) |

**Always** Scroll the caret back into view whenever input would position it off the screen.

**/Tty/Checkpoint_frequency**
If set to **0** (the standard default) no checkpointing is done. For any value greater than zero, a checkpoint is made each time the indicated number editing operations has been performed since the last checkpoint. Each character typed, each **Paste**, and each **Cut** counts as an editing operation. At each checkpoint, an updated copy of the log is saved in a file with a name that is constructed by appending two percent signs (**%%**) to the name of the log file. By default, the log file has a name of the form **/tmp/tty.txt.***pid* (*pid* is the process ID number of **cmdtool**); the corresponding checkpoint file has a name of the form **/tmp/tty.txt.***nnnnnn***%%**.

**/Tty/Text_wraparound_size**
If set to **0** (the standard default) no wrap-around takes place; the log file grows without a specified limit. For values greater than zero, wrap-around occurs whenever the indicated number of characters have been written to the log since the last wrap-around. Characters that are pushed over the top are replaced by the message:

***** Text is lost because the maximum edit log size has been exceeded. *****

**/Text/Edit_back_char**
Set the character for erasing to the left of the caret. Note: in **cmdtool**, the '**stty erase**' command has no effect on **cmdtool**. Text-based tools refer only to the defaults database key settings. The default is DELETE.

**/Text/Edit_back_word**
Set the character for erasing the word to the left of the caret. The standard default is CTRL-W.

**/Text/Edit_back_line**
Set the character for erasing all characters to the left of the caret. Note: '**stty kill**' has no effect on **cmdtool**. The standard default is CTRL-U.

**The Command Subwindow**

The command subwindow is based on the text facility, which is described in *SunView 1 Beginner's Guide*. It uses the same pop-up menu as the text facility, but with an additional pull-right '**Cmd Modes**' menu, which contains the '**Enable Editing**' and '**Disable Scrolling**' items.

Command subwindows support cursor motions, using a new **/etc/termcap** entry called **sun-cmd**. Command subwindows automatically set the TERM environment variable to **sun-cmd**. So, if you **rlogin**(1C) to a machine that does not have an entry for **sun-cmd** in its **/etc/termcap** file, the error message '**Type sun-cmd unknown**' results. To rectify this, type the command '**set TERM=sun**'. Programs written using the **curses**(3X) or **curses**(3V) library packages will work in a command subwindow, but programs hard-coded for **sun**-type terminals may not. When supporting a program that performs cursor motions, the command subwindow automatically takes on the characteristics of a tty subwindow (as with **shelltool**(1)). When that program terminates or sleeps, the full command subwindow functionality is restored.

**cmdtool** supports programs that use CBREAK and NO ECHO terminal modes. This support is normally invisible to the user. However, programs that use RAW mode, such as **rlogin**(1C) and **script**(1), inhibit command-line editing with the mouse. In this case, however, tty-style ERASE, word-kill and line-kill characters can still be used to edit the current command line.

**The Command Subwindow Menu**

**Copy, then Paste** When there is a current selection, the entire menu item is active. The selection is copied both to the clipboard and to the location pointed to by the caret. When there is no selection, but there is text on the clipboard, only **Paste** is active. In this case, the contents of the clipboard are copied to the caret. When there is no selection and the clipboard is empty, this item is inactive. '**Copy then Paste**' is a generic text menu item. Refer to **textedit**(1) information about other generic text menu items.

**Enable Edit**
**Disable Edit** Toggle to allow or disallow editing on the log.

**Disable Scrolling**
**Enable Scrolling** Toggle between a scrollable, editable window, or a display that supports cursor motions. Note: for well-behaved programs (such as **vi**(1)) this switching is performed automatically (so this menu item is seldom needed).

Accelerators

Text facility accelerators that are especially useful in command subwindows are described here. See **textedit**(1) for more information.

CTRL-RETURN Position the caret at the bottom, and scroll it into view as determined by **/Text/Lower_context**.

META-P Choose the '**Copy, then Paste**' menu item.

CAPS-lock
F1 Toggle between all-upper-case keyboard input, and mixed-case.

FILES

**/tmp/tty.txt.***pid* log file
**/.textswrc**
**/.ttyswrc**
**usr/lib/.text_extras_menu**
**/etc/termcap**
**/usr/bin/sh**

SEE ALSO

**defaultsedit**(1), **rlogin**(1C), **script**(1), **shelltool**(1), **sunview**(1), **textedit**(1), **vi**(1), **curses**(3V), **curses**(3X)

*Installing the SunOS*
*SunView 1 Beginner's Guide*

BUGS

Full terminal emulation is not complete. Some manifestations of this deficiency are:

- File completion in the C shell does not work.
- Enhanced display of text is not supported.

## NAME

cmp – perform a byte-by-byte comparison of two files

## SYNOPSIS

**cmp** [ **–ls** ] *filename1 filename2* [ *skip1* ] [ *skip2* ]

## DESCRIPTION

**cmp** compares *filename1* and *filename2*. If *filename1* is '–', the standard input is used. With no options, **cmp** makes no comment if the files are the same; if they differ, it reports the byte and line number at which the difference occurred, or, that one file is an initial subsequence of the other. *skip1* and *skip2* are initial byte offsets into *filename1* and *filename2* respectively, and may be either octal or decimal; a leading **0** denotes octal.

## OPTIONS

**–l** Print the byte number (in decimal) and the differing bytes (in octal) for all differences between the two files.

**–s** Silent. Print nothing for differing files; set exit codes only.

## SEE ALSO

**comm**(1), **diff**(1)

## DIAGNOSTICS

Exit code **0** is returned for identical files, **1** for different files, and **2** for an inaccessible or missing argument, or a system error.

NAME

col – filter reverse paper motions from nroff output for display on a terminal

SYNOPSIS

**col** [ **–bfhp** ]

SYSTEM V SYNOPSIS

**/usr/5bin/col** [ **–bfpx** ]

DESCRIPTION

**col** copies the standard input to the standard output and performs line overlays implied by reverse LINEFEED characters (ESC–7 in ASCII) and by forward and reverse half LINEFEED characters (ESC–9 and ESC–8). **col** is particularly useful for filtering multicolumn output made with the **.rt** command of **nroff**(1), and output resulting from use of the **tbl**(1) preprocessor.

The control characters SO (ASCII code 017), and SI (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

All control characters are removed from the input except SPACE, BACKSPACE, TAB, RETURN, NEWLINE, ESC (033) followed by one of 7, 8, 9, SI , SO , and VT (013). This last character is an alternate form of full reverse LINEFEED, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

SYSTEM V DESCRIPTION

The System V version of **col** converts SPACE to TAB characters by default.

OPTIONS

**–b** The output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

**–f** Fine. Although **col** accepts half line motions in its input, it normally does not produce them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. The **–f** option suppresses this treatment. In this case the output from **col** may contain forward half LINEFEED characters (ESC–9), but will still never contain either kind of reverse line motion.

**–h** Convert strings of blanks to TAB characters to decrease the printing time.

**–p** Pass escape-sequences that **col** does not know about to the output, rather than stripping them out. This option is highly discouraged unless you are fully aware of the position of the escape sequences within the text.

SYSTEM V OPTIONS

**–x** Suppress converting SPACE characters to TAB characters.

SEE ALSO

**nroff**(1), **tbl**(1), **troff**(1)

BUGS

**col** cannot back up more than 128 lines.

At most 1600 characters, including BACKSPACE characters, are allowed on a line.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

colcrt – filter nroff output for a terminal lacking overstrike capability

SYNOPSIS

**colcrt** [ – ] [ –2 ] [ *filename* ... ]

DESCRIPTION

**colcrt** provides virtual half-line and reverse line feed sequences for terminals without such capability, and on which overstriking is destructive. Half-line characters and underlining (changed to dashing '–') are placed on new lines in between the normal output lines.

OPTIONS

– Suppress all underlining — especially useful for previewing *allboxed* tables from **tbl**(1).

–2 Print all half-lines, effectively double spacing the output. Normally, a minimal space output format is used which suppresses empty lines. **colcrt** never suppresses two consecutive empty lines, however. The –2 option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

EXAMPLE

A typical use of **colcrt** would be

**tbl exum2.n | nroff –ms | colcrt – | more**

SEE ALSO

**col**(1V), **more**(1), **nroff**(1), **tbl**(1), **troff**(1), **ul**(1)

BUGS

Can't back up more than 102 lines.

General overstriking is lost; as a special case '|' overstruck with '–' or underline becomes '+'.

Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.

## NAME

coloredit – alter color map segment

## SYNOPSIS

**coloredit**

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

**coloredit** is a SunView application for altering the colors of objects (windows) selected with the cursor. These colors may be manipulated in three ways:

- select color name from a list
- specify the hue/saturation/value for a color
- specify the red/green/blue levels for a color

## USAGE

### Window

The **coloredit** window consists of four sections.

colorlist — An interactive window that lists the available colors for selection. This list can be scrolled up and down with a scrollbar, and comprises the contents of the **.rgb** file. Select a color and the corresponding RGB and HSV value appears in the colorbar section. **coloredit** first searches the working directory for **.rgb**, your home directory, and finally, **/usr/lib** this file.

colorbars — An interactive window with two sets of three sliding bars. One set is for defining the hue saturation value of the color desired. The second set is for defining the red-green-blue ratios. The color can be specified using either set of sliding bars.

palette — A display subwindow divided horizontally into two equal parts. The top part shows the background color; the bottom, the foreground color. If the object has more than 2 colors, all the colors are shown. The background color is color Index #0. The color with the highest Index Number is the foreground color.

logo — A display subwindow containing the Sun logo in the currently-selected color.

### Command Buttons

The set of command buttons in the colorbars subwindow is as follows.

**Grab** The next object selected will have its color map segment colors displayed in the *coloredit* window. These colors may then be manipulated.

**Release** Disassociates the colormap segment and the last object with which those colors were associated. The window returns to its default state.

**Undo** Returns the colormap segment to the original colors that the object being colored had when coloring began. Returns the object to its original colors.

## FILES

**./.rgb**
**/.rgb**
**/usr/lib/.rgb**

## SEE ALSO

**sunview**(1)

NAME

colrm – remove characters from specified columns within each line

SYNOPSIS

**colrm** [ *startcol* [ *endcol* ] ]

DESCRIPTION

**colrm** removes selected columns from a text file. The text is is taken from standard input and copied to the standard output with the specified columns removed.

If only *startcol* is specified, the columns of each line are removed starting with *startcol* and extending to the end of the line. If both *startcol* and *endcol* are specified, all columns between *startcol* and *endcol*, inclusive, are removed.

Column numbering starts with column 1.

SEE ALSO

**expand**(1)

NAME

comb – combine SCCS deltas

SYNOPSIS

**/usr/sccs/comb** [ **–os** ] [ **–c***list* ] [ **–p***SID* ] *filename* ...

DESCRIPTION

**comb** generates a shell procedure (see **sh**(1)) that can be used to reconstruct the given SCCS files. If a directory is named, **comb** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with 's.') and unreadable files are silently ignored. If a name of '-' is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

OPTIONS

Options are explained as though only one named file is to be processed, but the effects of any option apply independently to each named file.

**–o** For each '**get –e**' generated, the reconstructed file is accessed at the release of the delta to be created. In the absence of the **–o** option, the reconstructed file is accessed at the most recent ancestor. Use of the **–o** option may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

**–s** Generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

**example%100** * (*original – combined*) / *original*

It is recommended that before any SCCS files are actually combined, you should use this option to determine exactly how much space is saved by the combining process.

**–c***list* A *list* of deltas to be preserved. All other deltas are discarded. See **get**(1) for the syntax of a *list*.

**–p***SID* The SCCS ID string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

If no options are specified, **comb** preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

| | |
|---|---|
| **s.COMB** | reconstructed SCCS file |
| **comb?????** | temporary |

SEE ALSO

**admin**(1), **delta**(1), **get**(1), **help**(1), **prs**(1), **sccs**(1), **sh**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

DIAGNOSTICS

Use **help**(1) for explanations.

BUGS

**comb** may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

**NAME**

comm – display lines in common, and lines not in common, between two sorted lists

**SYNOPSIS**

**comm** [ **–1** | **–2** | **–3** | **–12** | **–13** | **–23** ] *filename1 filename2*

**DESCRIPTION**

**comm** reads *filename1* and *filename2*, which should be ordered in ASCII collating sequence (see **sort**(1V)), and produces three-column output when no options are specified:

- Column 1 contains lines that occur only in *filename1*.
- Column 2 contains lines only in *filename2*.
- Column 3 contains lines common to both files.

The filename '**-**' means the standard input.

**OPTIONS**

The following options can be used to suppress the indicated columns from display. You can specify '**–123**', but doing so suppresses all output.

**–1** Suppress column 1; omit lines only in *filename1*.

**–2** Suppress column 2; omit lines only in *filename2*.

**–3** Suppress column 3; omit lines common to both files.

**–12** Suppress columns 1 and 2; only show lines common to both files.

**–13** Suppress columns 1 and 3; only show lines in *filename2*.

**–23** Suppress columns 2 and 3; only show lines in *filename1*.

**SEE ALSO**

**cmp**(1), **diff**(1), **uniq**(1), **sort**(1V)

**BUGS**

The options *suppress*, rather than *select* the columns you indicate.

## NAME

compress, uncompress, zcat – compress or expand files, display expanded contents

## SYNOPSIS

**compress** [ **–cfv** ] [ **–b** *bits* ] [ *filename*... ]

**uncompress** [ **–cv** ] [ *filename*... ]

**zcat** [ *filename*... ]

## DESCRIPTION

**compress** reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with the extension **.Z**, while keeping the same ownership modes, as well as access and modification times. If no files are specified, the standard input is compressed to the standard output.

The amount of compression obtained depends on the size of the input, the number of *bits* per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50–60%. Compression is generally much better than that achieved by Huffman coding (as used in **pack**(1)), or adaptive Huffman coding (**oldcompact**(1)), and takes less time to compute. The *bits* parameter specified during compression is encoded within the compressed file, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is subsequently allowed.

Compressed files can be restored to their original form using **uncompress**.

**zcat** produces uncompressed output on the standard output, but leaves the compressed **.Z** file intact.

## OPTIONS

**–c** Write to the standard output; no files are changed. The nondestructive behavior of **zcat** is identical to that of '**uncompress –c**'.

**–f** Force compression, even if the file does not actually shrink, or the corresponding **.Z** file already exists. Except when running in the background (under **/usr/bin/sh**), if **–f** is not given, prompt to verify whether an existing **.Z** file should be overwritten.

**–v** Verbose. Display the percentage reduction for each file compressed.

**–b** *bits* Set the upper limit (in bits) for common substring codes. *bits* must be between 9 and 16 (16 is the default).

## FILES

**/usr/bin/sh**

## SEE ALSO

**ln**(1), **oldcompact**(1), **pack**(1)

*A Technique for High Performance Data Compression*, Terry A. Welch, *IEEE Computer*, vol. 17, no. 6 (June 1984), pp. 8-19.

## DIAGNOSTICS

Exit status is normally 0. If the last file was not compressed because it became larger, the status is 2. If an error occurs, exit status is 1.

**Usage: compress [–fvc] [–b maxbits]** [*filename*...]
Invalid options were specified on the command line.

**Missing maxbits**
Maxbits must follow **–b** .

*filename*: **not in compressed format**
The file specified to **uncompress** has not been compressed.

*filename*: **compressed with** *xx***bits, can only handle** *yy***bits**
*filename* was compressed by a program that could deal with more *bits* than the compress code on this machine. Recompress the file with smaller *bits*.

*filename*: **already has .Z suffix -- no change**
The file is assumed to be already compressed. Rename the file and try again.

*filename*: **already exists; do you wish to overwrite (y or n)?**
Respond **y** if you want the output file to be replaced; **n** if not.

**uncompress: corrupt input**
A SIGSEGV violation was detected, which usually means that the input file is corrupted.

**Compression:** *xx.xx* **%**
Percentage of the input saved by compression. (Relevant only for **–v**.)

**-- not a regular file: unchanged**
When the input file is not a regular file, (such as a directory), it is left unaltered.

**-- has** *xx* **other links: unchanged**
The input file has links; it is left unchanged. See **ln**(1) for more information.

**-- file unchanged**
No savings are achieved by compression. The input remains uncompressed.

## BUGS

Although compressed files are compatible between machines with large memory, **–b12** should be used for file transfer to architectures with a small process data space (64KB or less).

**compress** should be more flexible about the existence of the **.Z** suffix.

NAME

cp – copy files

SYNOPSIS

**cp** [ **–ip** ] *filename1 filename2*
**cp –r** [ **–ip** ] *directory1 directory2*
**cp** [ **–iprR** ] *filename ... directory*

DESCRIPTION

**cp** copies the contents of *filename1* onto *filename2*. The mode and owner of *filename2* are preserved if it already existed; the mode of the source file is used otherwise. If *filename1* is a symbolic link, or a duplicate hard link, the contents of the file that the link refers to are copied; links are not preserved.

In the second form, **cp** recursively copies *directory1*, along with its contents and subdirectories, to *directory2*. If *directory2* does *not* exist, **cp** creates it and duplicates the files and subdirectories of *directory1* within it. If *directory2* does exist, **cp** makes a copy of the *directory1* directory within *directory2* (as a subdirectory), along with its files and subdirectories.

In the third form, each *filename* is copied to the indicated *directory*; the basename of the copy corresponds to that of the original. The destination *directory* must already exist for the copy to succeed.

**cp** refuses to copy a file onto itself.

OPTIONS

**–i** Interactive. Prompt for confirmation whenever the copy would overwrite an existing file. A **y** in answer confirms that the copy should proceed. Any other answer prevents **cp** from overwriting the file.

**–p** Preserve. Duplicate not only the contents of the original file or directory, but also the modification time and permission modes.

**–r**
**–R** Recursive. If any of the source files are directories, copy the directory along with its files (including any subdirectories and their files); the destination must be a directory.

EXAMPLES

To copy a file:

```
example% cp goodies goodies.old
example% ls goodies*
goodies goodies.old
```

To copy a directory, first to a new, and then to an existing destination directory:

```
example% ls /bkup
/usr/example/fred/bkup not found
example% cp –r /src /bkup
example% ls –R /bkup
x.c y.c z.sh
example% cp –r /src /bkup
example% ls –R /bkup
src x.c y.c z.sh

src:
x.c y.c z.sh
```

BEWARE of a recursive copy like this:

```
example% cp –r /src /src/bkup
```

which keeps copying files until it fills the entire file system.

To copy a list of files to a destination directory:

```
example% cp /src/* /tmp
```

SEE ALSO

**cat**(1V), **ln**(1), **mv**(1), **pr**(1V), **rcp**(1C), **tar**(1)

BUGS

**cp** copies the contents of files pointed to by symbolic links. It does *not* copy the symbolic link itself. This can lead to inconsistencies when directory hierarchies are replicated. Filenames that were linked in the original hierarchy are no longer linked in the replica. This is also true for files with multiple hard links. See **ln**(1) for details about symbolic links and hard links. You can preserve links in replicated hierarchies by using **tar**(1) to copy them.

NAME

cpio – copy file archives in and out

SYNOPSIS

**cpio –o** [ **aBcv** ]
**cpio –i** [ **bcdfmrsStuv6** ] [ *patterns* ]
**cpio –p** [ **adlmruv** ] *directory*

DESCRIPTION

**cpio** copies files in to and out from a **cpio** copy archive. The archive (built by '**cpio –o**') contains pathname and status information, along with the contents of one or more archived files.

OPTIONS

**–o** Copy out an archive. Read the standard input for a list of pathnames, then copy the named files to the standard output in archive form — including pathname and status information.

**a** Reset the access times of input files after they have been copied.

**B** Input/output is to be blocked at 5120 bytes to the record. This does not apply to the *pass* option. This option is only meaningful with data directed to or from raw magnetic devices, such as '**/dev/rmt***?*'.

**c** Write *header* information in ASCII character form for portability.

**v** Verbose. A list of filenames is displayed. When used with the **t** option, the table of contents looks like the output of an '**ls –l**' command (see ls(1V)).

**–i** Copy in an archive. Read in an archive from the standard input and extract files with names matching filename substitution *patterns*, supplied as arguments.

*patterns* are similar to those in **sh**(1) or **csh**(1), save that within **cpio**, the metacharacters '**?**', '*****' and '[ ... ]' also match the '/' (slash) character. If no *patterns* are specified, the default is * (select all files).

**b** Swap both bytes and half-words after reading in data.

**d** Create directories as needed.

**f** Copy in all files except those matching *patterns*.

**m** Retain previous file modification time. This option is ineffective on directories that are being copied.

**r** Interactively rename files. If the user types a null line, the file is skipped.

**s** Swap bytes after reading in data.

**S** Swap halfwords after reading in data.

**t** Print a table of contents of the input archive. No files are created.

**u** Copy unconditionally. Normally, an older file will not replace a newer file with the same name.

**6** Process UNIX Version-6 files.

**–p** One pass. Copy in and out in a single operation. Destination pathnames are interpreted relative to the named *directory*.

**l** Whenever possible, link files rather than copying them.

EXAMPLES

To copy the contents of a directory into an archive:

**example% ls | cpio –o > /dev/mt0**

To duplicate the **olddir** directory hierarchy in the **newdir** directory:

**example% cd olddir**
**example% find . –depth –print | cpio –pdl newdir**

The trivial case

**example% find . –depth –print | cpio –oB >/dev/rmt0**

can be handled more efficiently by:

**example% find . –cpio /dev/rmt/0m**

**cpio** archive tapes from other sites may have bytes swapped within the archive. Although the **–is** option only swaps the data bytes and not those in the header **cpio** recognizes tapes like this and swaps the bytes in the header automatically.

SEE ALSO

**ar**(1V), **csh**(1), **find**(1), ls(1V), **sh**(1), **tar**(1), **cpio**(5)

BUGS

Pathnames are restricted to 128 characters. If there are too many unique linked files, **cpio** runs out of memory and linking information is lost thereafter. Only the super-user can copy special files.

NAME

cpp – the C language preprocessor

SYNOPSIS

**/usr/lib/cpp** [ **–BCHMpPRT** ] [ **–undef** ] [ **–D***name* ] [ **–D***name=def* ] [ **–I***dir* ] [ **–U name** ] [ **–Y***dir* ] [ *ifile* [ *ofile* ] ]

DESCRIPTION

**cpp** is the C language preprocessor. It is invoked as the first pass of any C compilation started with the **cc**(1V) command; however, **cpp** can also used as a first-pass preprocessor for other Sun compilers.

Although **cpp** can be used as a macro processor, this is not normally recommended, as its output is geared toward that which would be acceptable as input to a compiler's second pass. Thus, the preferred way to invoke **cpp** is through the **cc**(1V) command, or some other compilation command. For general-purpose macro-processing, see **m4**(1V), and the chapter on **m4** in *Programming Utilities and Libraries*

**cpp** optionally accepts two filenames as arguments. *ifile* and *ofile* are, respectively, the input and output files for the preprocessor. They default to the standard input and standard output.

OPTIONS

**–B** Support the C++ comment indicator //. With this indicator everything on the line after the // is treated as a comment.

**–C** Pass all comments (except those that appear on **cpp** directive lines) through the preprocessor. By default, **cpp** strips out C-style comments.

**–H** Print the pathnames of included files, one per line on the standard error.

**–M** Generate a list of makefile dependencies and write them to the standard output. This list indicates that the object file which would be generated from the input file depends on the input file as well as the include files referenced.

**–p** Use only the first eight characters to distinguish preprocessor symbols, and issue a warning if extra tokens appear at the end of a line containing a directive.

**–P** Preprocess the input without producing the line control information used by the next pass of the C compiler.

**–R** Allow recursive macros.

**–T** Use only the first eight characters for distinguishing different preprocessor names. This option is included for backward compatibility with systems which always use only the first eight characters.

**–undef** Remove initial definitions for all predefined symbols.

**–D***name*

Define *name* as 1 (one). This is the same as if a **–D***name*=**1** option appeared on the **cpp** command line, or as if a

**#define** *name* **1**

line appeared in the source file that **cpp** is processing.

**–D***name*=*def*

Define *name* as if by a **#define** directive. This is the same as if a

**#define** *name def*

line appeared in the source file that **cpp** is processing. The **–D** option has lower precedence than the **–U** option. That is, if the same name is used in both a **–U** option and a **–D** option, the name will be undefined regardless of the order of the options.

**–I***dir* Insert *dir* into the search path for **#include** files with names beginning with '/'. *dir* is inserted ahead of the standard list of "include" directories. Thus, **#include** files with names enclosed in double-quotes (") are searched for first in the directory of the file with the **#include** line, then in directories named with **–I** options, and lastly, in directories from the standard list. For **#include**

files with names enclosed in angle-brackets (< >), the directory of the file with the **#include** line is not searched. See **Details** below for exact details of this search order.

*–Uname*

Remove any initial definition of *name*, where *name* is a symbol that is predefined by a particular preprocessor. Here is a partial list of symbols that may be predefined, depending upon the architecture of the system:

| | |
|---|---|
| Operating System: | **ibm**, **gcos**, **os**, **tss** and **unix** |
| Hardware: | **interdata**, **pdp11**, **u370**, **u3b**, **u3b2**, **u3b5**, **u3b15**, **u3b20d**, **vax**, **m68k** (or alternatively, **mc68000**), **M68010** (or **mc68010**), **M68020** (or **mc68020**), **ns32000**, **iAPX286**, **i386**, **sparc** , and **sun** |
| UNIX system variant: | **RES**, and **RT** |
| The **lint**(1V) command: | **lint** |

The symbols **sun** and **unix** are defined for all Sun systems, as is the value returned by the **mach** command. For a Sun-4 system, this value would be **sparc**. In addition, **mc68000** is defined for Sun-2 and Sun-3 systems.

*–Ydir* Use directory *dir* in place of the standard list of directories when searching for **#include** files.

## USAGE

### Directives

All **cpp** directives start with a pound-sign (#) as the first character on a line. White space (SPACE or TAB characters) can appear after the initial # for proper indentation.

**#define** *name token-string*

Replace subsequent instances of *name* with *token-string*.

**#define** *name*(*arg* [, *arg*] ... ) *token-string*

There can be no space between *name* and the '('. Replace subsequent instances of *name*, followed by a parenthesized list of arguments, with *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, **cpp** re-starts its scan for names to expand at the beginning of the newly created *token-string*.

**#undef** *name*

Remove any definition for the symbol *name*. No additional tokens are permitted on the directive line after *name*.

**#include** "*filename*"
**#include** <*filename*>

Read in the contents of *filename* at this location. This data is processed by **cpp** as if it were part of the current file. When the <*filename*> notation is used, *filename* is only searched for in the standard "include" directories. See the **–I** and **–Y** options above for more detail. No additional tokens are permitted on the directive line after the final " or >.

**#line** *integer-constant* "*filename*"

Generate line control information for the next pass of the C compiler. *integer-constant* is interpreted as the line number of the next line and *filename* is interpreted as the file from where it comes. If "*filename*" is not given, the current filename is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

**#if** *constant-expression*

Subsequent lines up to the matching **#else**, **#elif** , or **#endif** directive, appear in the output only if *constant-expression* yields a nonzero value. All binary non-assignment C operators, including '**&&**', '**||**', and '**,**', are legal in *constant-expression*. The '**?:**' operator, and the unary '**–**', '**!**', and ' ' operators, are also legal in *constant-expression*.

The precedence of these operators is the same as that for C. In addition, the unary operator **defined**, can be used in *constant-expression* in these two forms: '**defined** ( *name* )' or '**defined** *name*'. This allows the effect of **#ifdef** and **#ifndef** directives (described below) in the **#if** directive. Only these operators, integer constants, and names that are known by **cpp** should be used within *constant-expression*. In particular, the **sizeof** operator is not available.

**#ifdef** *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has been defined, either with a **#define** directive or a **–D** option, and in the absence of an intervening **#undef** directive. No additional tokens are permitted on the directive line after *name*.

**#ifndef** *name*

Subsequent lines up to the matching **#else**, **#elif**, or **#endif** appear in the output only if *name* has *not* been defined, or if its definition has been removed with an **#undef** directive. No additional tokens are permitted on the directive line after *name* .

**#elif** *constant-expression*

Any number of **#elif** directives may appear between an **#if**, **#ifdef**, or **#ifndef** directive and a matching **#else** or **#endif** directive. The lines following the **#elif** directive appear in the output only if all of the following conditions hold:

- The *constant-expression* in the preceding **#if** directive evaluated to zero, the *name* in the preceding **#ifdef** is not defined, or the *name* in the preceding **#ifndef** directive *was* defined.
- The *constant-expression* in all intervening **#elif** directives evaluated to zero.
- The current *constant-expression* evaluates to non-zero.

If the *constant-expression* evaluates to non-zero, subsequent **#elif** and **#else** directives are ignored up to the matching **#endif**. Any *constant-expression* allowed in an **#if** directive is allowed in an **#elif** directive.

**#else** This inverts the sense of the conditional directive otherwise in effect. If the preceding conditional would indicate that lines are to be included, then lines between the **#else** and the matching **#endif** are ignored. If the preceding conditional indicates that lines would be ignored, subsequent lines are included in the output. Conditional directives and corresponding **#else** directives can be nested.

**#endif** End a section of lines begun by one of the conditional directives **#if**, **#ifdef**, or **#ifndef**. (Each such directive must have a matching **#endif**).

**Macros**

Formal parameters for macros are recognized in **#define** directive bodies, even when they occur inside character constants and quoted strings. For instance, the output from:

```
#define abc(a) |\a|
abc(xyz)
```

is the seven characters '|\xyz|' (SPACE, vertical-bar, backquote, x, y, z, vertical-bar). Macro names are not recognized within character constants or quoted strings during the regular scan. Thus:

```
#define abc xyz
printf("abc");
```

does not expand **abc** in the second line, since it is inside a quoted string that is not part of a **#define** macro definition.

Macros are not expanded while processing a **#define** or **#undef**. Thus:

```
#define abc zingo
#define xyz abc
#undef abc
xyz
```

produces **abc**. The token appearing immediately after an **#ifdef** or **#ifndef** is not expanded.

Macros are not expanded during the scan which determines the actual parameters to another macro call. Thus:

```
#define reverse(first,second)second first
#define greeting hello
reverse(greeting,
#define greeting goodbye
)
```

produces ' **goodbye** '.

**Output**

Output consists of a copy of the input file, with modifications, plus lines of the form:

*#lineno " filename " "level "*

indicating the original source line number and filename of the following output line and whether this is the first such line after an include file has been entered (*level*=1), the first such line after an include file has been exited (*level*=2), or any other such line (*level* is empty).

**Details**

*Directory Search Order*

**#include** files is:

1. The directory of the file that contains the **#include** request (that is, **#include** is relative to the file being scanned when the request is made).
2. The directories specified by **–I** options, in left-to-right order.
3. The standard directory(s) (**/usr/include** on UNIX systems).

*Special Names*

Two special names are understood by **cpp**. The name **_ _LINE_ _** is defined as the current line number (a decimal integer) as known by **cpp**, and **_ _FILE_ _** is defined as the current filename (a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

*Newline Characters*

A NEWLINE character terminates a character constant or quoted string. An escaped NEWLINE (that is, a backslash immediately followed by a NEWLINE) may be used in the body of a **#define** statement to continue the definition onto the next line. The escaped NEWLINE is not included in the macro value.

*Comments*

Comments are removed (unless the **–C** option is used on the command line). Comments are also ignored, except that a comment terminates a token.

FILES

| | |
|---|---|
| **/usr/include** | standard directory for **#include** files |

SEE ALSO

**cc**(1V), **m4**(1V)

**m4** — *A Macro Processor* in *Programming Utilities and Libraries*

DIAGNOSTICS

The error messages produced by **cpp** are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When NEWLINE characters were found in argument lists for macros to be expanded, some previous versions of **cpp** put out the NEWLINE characters as they were found and expanded. The current version of **cpp** replaces them with SPACE characters.

Because the standard directory for included files may be different in different environments, this form of **#include** directive:

**#include <file.h>**

should be used, rather than one with an absolute path, like:

**#include "/usr/include/file.h"**

**cpp** warns about the use of the absolute pathname.

NAME

crontab – install, edit, remove or list a user's crontab file

SYNOPSIS

**crontab** [ *filename* ]
**crontab –e** [ *username* ]
**crontab –l** [ *username* ]
**crontab –r**

DESCRIPTION

**crontab** copies the specified file, or the standard input if no file is specified, into a directory that holds all users' **crontab** files. A user's **crontab** file lists commands that are to be executed on behalf of that user at specified times on specified dates; the format of these files is described in **crontab**(5).

If the file **/var/spool/cron/at.allow** exists, only users whose username appears in it can use **crontab**. If that file does *not* exist, however, **crontab** checks the **/var/spool/cron/at.deny** file to determine if the user should be denied the use of **crontab**. If neither file exists, only the super-user is allowed to submit a **crontab** job. If **at.allow** does not exist and **at.deny** exists and is empty, global usage is permitted. The allow/deny files consist of one user name per line.

OPTIONS

**–e** Make a copy of the current user's **crontab** file, or create an empty file if it does not exist, and edit that file. The **vi**(1) editor will be used unless the environment variable **VISUAL** or **EDITOR** indicates an alternate editor. When editing is complete, install the file as the user's **crontab** file if it was modified. If a *username* is given, the specified user's **crontab** file is edited, rather than the current user's **crontab** file; this may only be done by the super-user.

**–l** List the user's **crontab** file.

**–r** Remove the current user's **crontab** file from the **crontab** directory. If a *username* is given, the specified user's **crontab** file is removed, rather than the current user's **crontab** file; this may only be done by the super-user.

FILES

**/var/spool/cron** main cron directory
**/var/spool/cron/crontabs** spool area
**/var/spool/cron/at.allow** list of allowed users
**/var/spool/cron/at.deny** list of denied users

SEE ALSO

**sh**(1), **crontab**(5), **cron**(8)

WARNINGS

If you inadvertently enter the **crontab** command with no argument(s), do not attempt to get out by typing CTRL-D. This removes all entries in your **crontab** file. Instead, exit by typing your interrupt character (normally CTRL-C).

NAME

crypt – encode or decode a file

SYNOPSIS

**crypt** [ *password* ]

DESCRIPTION

**crypt** encrypts and decrypts the contents of a file. **crypt** reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, **crypt** demands a key from the terminal and turns off printing while the key is being typed in. **crypt** encrypts and decrypts with the same key:

**example% crypt key** <*clear.file* >*encrypted.file*
**example% crypt key** <*encrypted.file* | **pr**

will print the contents of *clear.file*.

Files encrypted by **crypt** are compatible with those treated by the editors **ed**(1), **ex**(1) and **vi**(1) in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or cleartext can become visible must be minimized.

**crypt** implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are widely known, thus **crypt** provides minimal security.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, that is, to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the **crypt** command, it is potentially visible to users executing **ps**(1) or a derivative command. To minimize this possibility, **crypt** takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of **crypt.**

FILES

**/dev/tty** for typed key

SEE ALSO

**des**(1), **ed**(1), **ex**(1), **ps**(1), **vi**(1), **makekey**(8)

RESTRICTIONS

This program is not available on software shipped outside the U.S.

NAME

csh – a shell (command interpreter) with a C-like syntax and advanced interactive features

SYNOPSIS

**csh** [ **–bcefinstvVxX** ] [ *argument*... ]

DESCRIPTION

**csh**, the C shell, is a command interpreter with a syntax reminiscent of C. It provides a number of convenient features for interactive use that are not available with the standard (Bourne) shell, including filename completion, command aliasing, history substitution, job control, and a number of built-in commands. As with the standard shell, the C shell provides variable, command and filename substitution.

**Initialization and Termination**

When first started, the C shell normally performs commands from the **.cshrc** file in your home directory, provided that it is readable and you either own it or your real group ID matches its group ID. If the shell is invoked with a name that starts with '–', as when started by **login**(1), the shell runs as a *login* shell. In this case, after executing commands from the **.cshrc** file, the shell executes commands from the **.login** file in your home directory; the same permission checks as those for **.cshrc** are applied to this file. Typically, the **.login** file contains commands to specify the terminal type and environment.

As a login shell terminates, it performs commands from the **.logout** file in your home directory; the same permission checks as those for **.cshrc** are applied to this file.

**Interactive Operation**

After startup processing is complete, an interactive C shell begins reading commands from the terminal, prompting with *hostname%* (or *hostname#* for the super-user). The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the history list and then parsed, as described under USAGE, below. Finally, the shell executes each command in the current line.

**Noninteractive Operation**

When running noninteractively, the shell does not prompt for input from the terminal. A noninteractive C shell can execute a command supplied as an *argument* on its command line, or interpret commands from a script.

OPTIONS

**–b** Force a "break" from option processing. Subsequent command-line arguments are not interpreted as C shell options. This allows the passing of options to a script without confusion. The shell does not run a set-user-ID script unless this option is present.

**–c** Read commands from the first filename *argument* (which must be present). Remaining arguments are placed in **argv**, the argument-list variable.

**–e** Exit if a command terminates abnormally or yields a nonzero exit status.

**–f** Fast start. Read neither the **.cshrc** file, nor the **.login** file (if a login shell) upon startup.

**–i** Forced interactive. Prompt for command-line input, even if the standard input does not appear to be a terminal (character-special device).

**–n** Parse (interpret), but do not execute commands. This option can be used to check C shell scripts for syntax errors.

**–s** Take commands from the standard input.

**–t** Read and execute a single command line. A backslash (\) can be used to escape each NEWLINE for continuation of the command line onto subsequent input lines.

**–v** Verbose. Set the **verbose** predefined variable; command input is echoed after history substitution (but before other substitutions) and before execution.

**–V** Set **verbose** before reading **.cshrc**.

**-x** Echo. Set the **echo** variable; echo commands after all substitutions and just before execution.

**-X** Set **echo** before reading **.cshrc**.

Except with the flags **-c**, **-i**, **-s** or **-t**, the first nonflag *argument* is taken to be the name of a command or script. It is passed as argument zero, and subsequent arguments are added to the argument list for that command or script.

USAGE

Refer to *Doing More with SunOS: Beginner's Guide* for tutorial information on how to use the various features of the C shell.

**Filename Completion**

When enabled by setting the variable **filec**, an interactive C shell can complete a partially typed filename or user name. When an unambiguous partial filename is followed by an ESC character on the terminal input line, the shell fills in the remaining characters of a matching filename from the working directory.

If a partial filename is followed by the EOF character (usually typed as **^D**), the shell lists all filenames that match. It then prompts once again, supplying the incomplete command line typed in so far.

When the last (partial) word begins with a tilde (~), the shell attempts completion with a user name, rather than a file in the working directory.

The terminal bell signals errors or multiple matches; this can be inhibited by setting the variable **nobeep**. You can exclude files with certain suffixes by listing those suffixes in the variable **fignore**. If, however, the only possible completion includes a suffix in the list, it is not ignored. **fignore** does not affect the listing of filenames by the EOF character.

**Lexical Structure**

The shell splits input lines into words at SPACE and TAB characters, except as noted below. The characters &, |, ;, <, >, (, and ) form separate words; if paired, the pairs form single words. These shell metacharacters can be made part of other words, and their special meaning can be suppressed by preceding them with a backslash (\). A NEWLINE preceded by a \ is equivalent to a SPACE.

In addition, a string enclosed in matched pairs of single-quotes ( ' ), double-quotes ( " ), or backquotes ( ` ), forms a partial word; metacharacters in such a string, including any SPACE or TAB characters, do not form separate words. Within pairs of backquote ( ` ) or double-quote ( " ) characters, a NEWLINE preceded by a backslash ( \ ) gives a true NEWLINE character. Additional functions of each type of quote are described, below, under **Variable Substitution**, **Command Substitution**, and **Filename Substitution**.

When the shell's input is not a terminal, the character # introduces a comment that continues to the end of the input line. Its special meaning is suppressed when preceded by a \ or enclosed in matching quotes.

**Command Line Parsing**

A *simple command* is composed of a sequence of words. The first word (that is not part of an I/O redirection) specifies the command to be executed. A simple command, or a set of simple commands separated by | or |& characters, forms a *pipeline*. With |, the standard output of the preceding command is redirected to the standard input of the command that follows. With |&, both the standard error and the standard output are redirected through the pipeline.

Pipelines can be separated by semicolons ( ; ), in which case they are executed sequentially. Pipelines that are separated by && or || form conditional sequences in which the execution of pipelines on the right depends upon the success or failure, respectively, of the pipeline on the left.

A pipeline or sequence can be enclosed within parentheses '( )' to form a simple command that can be a component in a pipeline or sequence.

A sequence of pipelines can be executed asynchronously, or "in the background" by appending an '&'; rather than waiting for the sequence to finish before issuing a prompt, the shell displays the job number (see **Job Control**, below) and associated process IDs, and prompts immediately.

**History Substitution**

History substitution allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments. Command lines are saved in the history list, the size of which is controlled by the **history** variable. The most recent command is retained in any case. A history substitution begins with a **!** (although you can change this with the **histchars** variable) and may occur anywhere on the command line; history substitutions do not nest. The **!** can be escaped with \ to suppress its special meaning.

Input lines containing history substitutions are echoed on the terminal after being expanded, but before any other substitutions take place or the command gets executed.

*Event Designators*

An event designator is a reference to a command-line entry in the history list.

| | |
|---|---|
| **!** | Start a history substitution, except when followed by a SPACE, TAB, NEWLINE, = or (. |
| **!!** | Refer to the previous command. By itself, this substitution repeats the previous command. |
| **!***n* | Refer to command-line *n*. |
| **!–***n* | Refer to the current command-line minus *n*. |
| **!***str* | Refer to the most recent command starting with *str*. |
| **!?***str*[**?**] | Refer to the most recent command containing *str*. |
| **!{...}** | Insulate a history reference from adjacent characters (if necessary). |

*Word Designators*

A **:** separates the event specification from the word designator. It can be omitted if the word designator begins with a **^**, **$**, *****, **–** or **%**. If the word is to be selected from the previous command, the second **!** character can be omitted from the event specification. For instance, **!!:1** and **!:1** both refer to the first word of the previous command, while **!!$** and **!$** both refer to the last word in the previous command. Word designators include:

| | |
|---|---|
| **#** | The entire command line typed so far. |
| **0** | The first input word (command). |
| ***n*** | The *n*'th argument. |
| **^** | The first argument, that is, **1**. |
| **$** | The last argument. |
| **%** | The word matched by (the most recent) **?***s* search. |
| ***x*–*y*** | A range of words; –*y* abbreviates **0**–*y*. |
| ***** | All the arguments, or a null value if there is just one word in the event. |
| ***x**** | Abbreviates *x*–**$**. |
| ***x*–** | Like *x** but omitting word **$**. |

*Modifiers*

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a **:**.

| | |
|---|---|
| **h** | Remove a trailing pathname component, leaving the head. |
| **r** | Remove a trailing suffix of the form '.*xxx*', leaving the basename. |
| **e** | Remove all but the suffix. |
| **s/***l*/*r*[/] | Substitute *r* for *l*. |
| **t** | Remove all leading pathname components, leaving the tail. |
| **&** | Repeat the previous substitution. |
| **g** | Apply the change to the first occurrence of a match in each word, by prefixing the above (for example, **g&**). |
| **p** | Print the new command but do not execute it. |
| **q** | Quote the substituted words, escaping further substitutions. |
| **x** | Like **q**, but break into words at each SPACE, TAB or NEWLINE. |

Unless preceded by a **g**, the modification is applied only to the first string that matches *l*; an error results if no string matches.

The left-hand side of substitutions are not regular expressions, but character strings. Any character can be used as the delimiter in place of /. A backslash quotes the delimiter character. The character &, in the right hand side, is replaced by the text from the left-hand-side. The & can be quoted with a backslash. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* from **!?***s*. You can omit the rightmost delimiter if a NEWLINE immediately follows *r*; the rightmost **?** in a context scan can similarly be omitted.

Without an event specification, a history reference refers either to the previous command, or to a previous history reference on the command line (if any).

*Quick Substitution*

^*l*^*r*[^] This is equivalent to the history substitution: **!:s**^*l*^*r*[^].

**Aliases**

The C shell maintains a list of aliases that you can create, display, and modify using the **alias** and **unalias** commands. The shell checks the first word in each command to see if it matches the name of an existing alias. If it does, the command is reprocessed with the alias definition replacing its name; the history substitution mechanism is made available as though that command were the previous input line. This allows history substitutions, escaped with a backslash in the definition, to be replaced with actual command-line arguments when the alias is used. If no history substitution is called for, the arguments remain unchanged.

Aliases can be nested. That is, an alias definition can contain the name of another alias. Nested aliases are expanded before any history substitutions is applied. This is useful in pipelines such as

**alias lm 'ls –l \!* | more'**

which when called, pipes the output of **ls**(1V) through **more**(1).

Execpt for the first word, the name of the alias may not appear in its definition, nor in any alias referred to by its definition. Such loops are detected, and cause an error message.

**I/O Redirection**

The following metacharacters indicate that the subsequent word is the name of a file to which the command's standard input, standard output, or standard error is redirected; this word is variable, command, and filename expanded separately from the rest of the command.

**<** Redirect the standard input.

**<<** *word*

Read the standard input, up to a line that is identical with *word*, and place the resulting lines in a temporary file. Unless *word* is escaped or quoted, variable and command substitutions are performed on these lines. Then, invoke the pipeline with the temporary file as its standard input. *word* is not subjected to variable, filename or command substitution, and each line is compared to it before any substitutions are performed by the shell.

**> >! >& >&!**

Redirect the standard output to a file. If the file does not exist, it is created. If it does exist, it is overwritten; its previous contents are lost.

When set, the variable **noclobber** prevents destruction of existing files. It also prevents redirection to terminals and **/dev/null**, unless one of the **!** forms is used. The **&** forms redirect both standard output and the the standard error (diagnostic output) to the file.

**>> >>& >>! >>&!**

Append the standard output. Like **>**, but places output at the end of the file rather than overwriting it. If **noclobber** is set, it is an error for the file not to exist, unless one of the **!** forms is used. The **&** forms append both the standard error and standard output to the file.

**Variable Substitution**

After an input line is aliased and parsed, and before each command is executed, variable substitution is performed. I/O redirections are recognized before variable expansion is applied, and are variable-expanded separately. Strings enclosed in backquotes (`), even when they contain variable references, are interpreted later (see **Command Substitution**). Otherwise, variable substitution is performed on the command name and argument list together.

The C shell maintains a set of *variables*, each of which is composed of a *name* and a *value*. A variable name consists of up to 20 letters and digits, and starts with a letter (the underscore is considered a letter). A variable's value is a space-separated list of zero or more words. A reference to a variable starts with a $, and replaces the words of that variable's value, by selected words from the value, or by information about the variable, as described below. Braces can be used to insulate the reference from subsequent characters, which might otherwise be interpreted as part of it.

Variable substitution can be suppressed by preceding the $ with a \, except within double-quotes where it always occurs. Within single-quotes, variable substitution is suppressed. A $ is escaped if followed by a SPACE, TAB or NEWLINE.

Variables can be created, displayed, or destroyed using the **set** and **unset** commands. Some variables are maintained or used by the shell. For instance, the **argv** variable contains an image of the shell's argument list. Of the variables used by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not.

Numerical values can be operated on as numbers (as with the @ built-in). With numeric operations, an empty value is considered to be zero; the second and subsequent words of multiword values are ignored. For instance, when the **verbose** variable is set to any value (including an empty value), command input is echoed on the terminal.

Command and filename substitution is subsequently applied to the words that result from the variable substitution, except when suppressed by double-quotes, when **noglob** is set (suppressing filename substitution), or when the reference is quoted with the **:q** modifier. Within double-quotes, a reference is expanded to form (a portion of) a quoted string; multiword values are expanded to a string with embedded SPACE characters. When the **:q** modifier is applied to the reference, it is expanded to a list of space-separated words, each of which is quoted to prevent subsequent command or filename substitutions.

Except as noted below, it is an error to refer to a variable that is not set.

*$var*

*${var}* These are replaced by words from the value of *var*, each separated by a SPACE. If *var* is an environment variable, its value is returned (but ':' modifiers and the other forms given below are not available).

*$var[index]*

*${var[index]}*

These select only the indicated words from the value of *var*. Variable substitution is applied to *index*, which may consist of (or result in) a either single number, two numbers separated by a '–', or an asterisk. Words are indexed starting from 1; a '*' selects all words. If the first number of a range is omitted (as with **$argv[–2]**), it defaults to 1. If the last number of a range is omitted (as with **$argv[1–]**), it defaults to *$#var* (the word count). It is not an error for a range to be empty if the second argument is omitted (or within range).

*$#name*

*${#name}*

These give the number of words in the variable.

**$0** This substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

**$*n***

**${*n*}** Equivalent to **$argv[*n*]**.

**$*** Equivalent to **$argv[*]**.

The modifiers **:h**, **:t**, **:r**, **:q** and **:x** can be applied (see **History Substitution**), as can **:gh**, **:gt** and **:gr**. If braces ( { } ) are used, then the modifiers must appear within the braces. The current implementation allows only one such modifier per expansion.

The following references may not be modified with **:** modifiers.

**$?*var***

**${?*var*}** Substitutes the string 1 if *var* is set or 0 if it is not set.

**$?0** Substitutes 1 if the current input filename is known, or 0 if it is not.

**$$** Substitute the process number of the (parent) shell.

**$<** Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a C shell script.

**Command and Filename Substitutions**

Command and filename substitutions are applied selectively to the arguments of built-in commands. Portions of expressions that are not evaluated are not expanded. For non-built-in commands, filename expansion of the command name is done separately from that of the argument list; expansion occurs in a subshell, after I/O redirection is performed.

**Command Substitution**

A command enclosed by backquotes (`...`) is performed by a subshell. Its standard output is broken into separate words at each SPACE, TAB and NEWLINE; null words are discarded. This text replaces the backquoted string on the current command line. Within double-quotes, only NEWLINE characters force new words; SPACE and TAB characters are preserved. However, a final NEWLINE is ignored. It is therefore possible for a command substitution to yield a partial word.

**Filename Substitution**

Unquoted words containing any of the characters *, ?, [ or {, or that begin with ~, are expanded (also known as *globbing*) to an alphabetically sorted list of filenames, as follows:

***** Match any (zero or more) characters.

**?** Match any single character.

**[...]** Match any single character in the enclosed list(s) or range(s). A list is a string of characters. A range is two characters separated by a minus-sign (–), and includes all the characters in between in the ASCII collating sequence (see **ascii**(7)).

**{*str*,*str*,...}**

Expand to each string (or filename-matching pattern) in the comma-separated list. Unlike the pattern-matching expressions above, the expansion of this construct is not sorted. For instance, **{b,a}** expands to **'b' 'a'**, (not **'a' 'b'**). As special cases, the characters { and }, along with the string { }, are passed undisturbed.

**~*[user]*** Your home directory, as indicated by the value of the variable **home**, or that of *user*, as indicated by the password entry for *user*.

Only the patterns *, ? and [...] imply pattern matching; an error results if no filename matches a pattern that contains them. The dot character (.), when it is the first character in a filename or pathname component, must be matched explicitly. The slash (/) must also be matched explicitly.

**Expressions and Operators**

A number of C shell built-in commands accept expressions, in which the operators are similar to those of C and have the same precedence. These expressions typically appear in the **@**, **exit**, **if**, **set** and **while** commands, and are often used to regulate the flow of control for executing commands. Components of an expression are separated by white space.

Null or missing values are considered 0. The result of all expressions are strings, which may represent decimal numbers.

The following C shell operators are grouped in order of precedence:

( ... ) grouping
~ one's complement
! logical negation
* / % multiplication, division, remainder (These are right associative, which can lead to unexpected results. Group combinations explicitly with parentheses.)
+ − addition, subtraction (also right associative)
<< >> bitwise shift left, bitwise shift right
< > <= >= less than, greater than, less than or equal to, greater than or equal to
== != =~ !~ equal to, not equal to, filename-substitution pattern match (described below), filename-substitution pattern mismatch
& bitwise AND
^ bitwise XOR (exclusive or)
| bitwise inclusive OR
&& logical AND
| | logical OR

The operators: ==, !=, =~, and !~ compare their arguments as strings; other operators use numbers. The oprators =~ and !~ each check whether or not a string to the left matches a filename substitution pattern on the right. This reduces the need for **switch** statements when pattern-matching between strings is all that is required.

Also available are file inquiries:

−r *file* Return true, or 1 if the user has read access. Otherwise it returns false, or 0.
−w *file* True if the user has write access.
−x *file* True if the user has execute permission (or search permission on a directory).
−e *file* True if *file* exists.
−o *file* True if the user owns *file*.
−z *file* True if *file* is of zero length (empty).
−f *file* True if *file* is a plain file.
−d *file* True if *file* is a directory.

If *file* does not exist or is inaccessible, then all inquiries return false.

An inquiry as to the success of a command is also available:

{ *cmd* } If *cmd* runs successfully, the expression evaluates to true, 1. Otherwise it evaluates to false 0. (Note that, conversely, *cmd* itself typically returns 0 when it runs successfully, or some other value if it encounters a problem. If you want to get at the status directly, use the value of the **status** variable rather than this expression).

**Control Flow**

The shell contains a number of commands to regulate the flow of control in scripts, and within limits, from the terminal. These commands operate by forcing the shell either to reread input (to *loop*), or to skip input under certain conditions (to *branch*).

Each occurrence of a **foreach, switch, while, if...then** and **else** built-in must appear as the first word on its own input line.

If the shell's input is not seekable and a loop is being read, that input is buffered. The shell performs seeks within the internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward **goto** commands will succeed on nonseekable inputs.)

**Command Execution**

If the command is a C shell built-in, the shell executes it directly. Otherwise, the shell searches for a file by that name with execute access. If the command-name contains a /, the shell takes it as a pathname, and searches for it. If the command-name does not contain a /, the shell attempts to resolve it to a pathname, searching each directory in the **path** variable for the command. To speed the search, the shell uses its hash table (see the **rehash** built-in) to eliminate directories that have no applicable files. This hashing can be disabled with the **–c** or **–t**, options, or the **unhash** built-in.

As a special case, if there is no / in the in the name of the script and there is an alias for the word **shell**, the expansion of the **shell** alias is prepended (without modification), to the command line. The system attempts to execute the first word of this special (late-occurring) alias, which should be a full pathname. Remaining words of the alias's definition, along with the text of the input line, are treated as arguments.

When a pathname is found that has proper execute permissions, the shell forks a new process and passes it, along with its arguments to the kernel (using the **execve**(2) system call). The kernel then attempts to overlay the new process with the desired program. If the file is an executable binary (in **a.out**(5), the kernel succeeds, and begins executing the new process. If the file is a text file, and the first line begins with **#!**, the next word is taken to be the pathname of a shell (or command) to interpret that script. Subsequent words on the first line are taken as options for that shell. The kernel invokes (overlays) the indicated shell, using the name of the script as an argument.

If neither of the above conditions holds, the kernel cannot overlay the file (the **execve**(2) call fails); the C shell then attempts to execute the file by spawning a new shell, as follows:

- If the first character of the file is a **#**, a C shell is invoked.
- Otherwise, a standard (Bourne) shell is invoked.

**Signal Handling**

The shell normally ignores QUIT signals. Background jobs are immune to signals generated from the keyboard, including HUPs. Other signals have the values that the C shell inherited from its environment. The shell's handling of interrupt and terminate signals within scripts can be controlled by the **onintr** built-in. Login shells catch the TERM signal; otherwise this signal is passed on to child processes. In no case are interrupts allowed when a login shell is reading the **.logout** file.

**Job Control**

The shell associates a numbered *job* with each command sequence, to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically **^Z**). When a command, or command sequence (semicolon separated list), is started in the background using the **&** metacharacter, the shell displays a line with the job number in brackets, and a list of associated process numbers:

```
[1] 1234
```

To see the current list of jobs, use the **jobs** built-in command. The job most recently stopped (or put into the background if none are stopped) is referred to as the *current* job, and is indicated with a **+**. The previous job is indicated with a **–**; when the current job is terminated or moved to the foreground, this job takes its place (becomes the new current job).

To manipulate jobs, refer to the **bg**, **fg**, **kill**, **stop** and **%** built-ins.

A reference to a job begins with a **%**. By itself, the percent-sign refers to the current job.

**%** **%+** **%%**
: The current job.

**%–**
: The previous job.

***%j***
: Refer to job *j* as in: '**kill –9** ***%j***'. *j* can be a job number, or a string that uniquely specifies the command-line by which it was started; '**fg %vi**' might bring a stopped **vi** job to the foreground, for instance.

***%?string***
: Specify the job for which the command-line uniquely contains *string*.

A job running in the background stops when it attempts to read from the terminal. Background jobs can normally produce output, but this can be suppressed using the '**stty tostop**' command.

**Status Reporting**

While running interactively, the shell tracks the status of each job and reports whenever a finishes or becomes blocked. It normally displays a message to this effect as it issues a prompt, so as to avoid disturbing the appearance of your input. When set, the **notify** variable indicates that the shell is to report status changes immediately. By default, the **notify** command marks the current process; after starting a background job, type **notify** to mark it.

**Built-In Commands**

Built-in commands are executed within the C shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

**:** Null command. This command is interpreted, but performs no action.

**alias** [*name* [*def*]]
Assign *def* to the alias *name*. *def* is a list of words that may contain escaped history-substitution metasyntax. *name* is not allowed to be **alias** or **unalias**. If *def* is omitted, the alias *name* is displayed along with its current definition. If both *name* and *def* are omitted, all aliases are displayed.

**bg** [*%job*] ...
Run the current or specified jobs in the background.

**break** Resume execution after the **end** of the nearest enclosing **foreach** or **while** loop. The remaining commands on the current line are executed. This allows multilevel breaks to be written as a list of **break** commands, all on one line.

**breaksw** Break from a **switch**, resuming after the **endsw**.

**case** *label*: A label in a **switch** statement.

**cd** [*dir*]
**chdir** [*dir*]
Change the shell's working directory to directory *dir*. If no argument is given, change to the home directory of the user. If *dir* is a relative pathname not found in the current directory, check for it in those directories listed in the **cdpath** variable. If *dir* is the name of a shell variable whose value starts with a /, change to the directory named by that value.

**continue** Continue execution of the nearest enclosing **while** or **foreach**.

**default:** Labels the default case in a **switch** statement. The default should come after all **case** labels. Any remaining commands on the command line are first executed.

**dirs** [**–l**]
Print the directory stack, most recent to the left; the first directory shown is the current directory. With the **–l** argument, produce an unabbreviated printout; use of the ~ notation is suppressed.

**echo** [**–n**] *list*
The words in *list* are written to the shell's standard output, separated by SPACE characters. The output is terminated with a NEWLINE unless the **–n** option is used.

**eval** *arg* ... Reads the arguments as input to the shell, and executes the resulting command(s). This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See **tset**(1) for an example of how to use **eval**.

**exec** *command*
Execute *command* in place of the current shell, which terminates.

**exit** [(*expr*)]

The shell exits, either with the value of the **status** variable, or with the value of the specified by the expression *expr*.

**fg** % [*job*]

Bring the current or specified *job* into the foreground.

**foreach** *var* (*wordlist*)

...

**end**

The variable *var* is successively set to each member of *wordlist*. The sequence of commands between this command and the matching **end** is executed for each new value of *var*. (Both **foreach** and **end** must appear alone on separate lines.)

The built-in command **continue** may be used to continue the loop prematurely and the built-in command **break** to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with **?** before any statements in the loop are executed.

**glob** *wordlist*

Perform filename expansion on *wordlist*. Like **echo**, but no \ escapes are recognized. Words are delimited by NULL characters in the output.

**goto** *label* The specified *label* is filename and command expanded to yield a label. The shell rewinds its input as much as possible and searches for a line of the form *label*: possibly preceded by SPACE or TAB characters. Execution continues after the indicated line. It is an error to jump to a label that occurs between a **while** or **for** built-in, and its corresponding **end**.

**hashstat** Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **execs**). An **exec** is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component that does not begin with a '/'.

**history** [–**hr**] [*n*]

Display the history list; if *n* is given, display only the *n* most recent events.

–**r** Reverse the order of printout to be most recent first rather than oldest first.

–**h** Display the history list without leading numbers. This is used to produce files suitable for sourcing using the –**h** option to *source*.

**if** (*expr*) *command*

If the specified expression evaluates to true, the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Note: I/O redirection occurs even if *expr* is false, when *command* is **not** executed (this is a bug).

**if** (*expr*) **then**

...

**else if** (*expr2*) **then**

...

**else**

...

**endif**

If *expr* is true, commands up to the first **else** are executed. Otherwise, if *expr2* is true, the commands between the **else if** and the second **else** are executed. Otherwise, commands between the **else** and the **endif** are executed. Any number of **else if** pairs are allowed, but only one **else**. Only one **endif** is needed, but it is required. The words **else** and **endif** must be the first nonwhite characters on a line. The **if** must appear alone on its input line or after an **else**.)

**jobs**[–**l**] List the active jobs under job control.

–**l** List process IDs, in addition to the normal information.

**kill** [*–sig*] [*pid*] [*%job* ] ...

**kill –l** Send the TERM (terminate) signal, by default, or the signal specified, to the specified process ID, the *job* indicated, or the current *job*. Signals are either given by number or by name. There is no default. Typing **kill** does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

**–l** List the signal names that can be sent.

**limit** [**–h**] [*resource* [*max-use*] ]

Limit the consumption by the current process or any process it spawns, each not to exceed *max-use* on the specified *resource*. If *max-use* is omitted, print the current limit; if *resource* is omitted, display all limits.

**–h** Use hard limits instead of the current limits. Hard limits impose a ceiling on the values of the current limits. Only the super-user may raise the hard limits.

*resource* is one of:

| | |
|---|---|
| **cputime** | Maximum CPU seconds per process. |
| **filesize** | Largest single file allowed. |
| **datasize** | Maximum data size (including stack) for the process. |
| **stacksize** | Maximum stack size for the process. |
| **coredumpsize** | Maximum size of a core dump (file). |

*max-use* is a number, with an optional scaling factor, as follows:

| | |
|---|---|
| *n***h** | Hours (for **cputime**). |
| *n***k** | *n* kilobytes. This is the default for all but **cputime**. |
| *n***m** | *n* megabytes or minutes (for **cputime**). |
| *mm*:*ss* | Minutes and seconds (for **cputime**). |

**login** [*username* | **–p**]

Terminate a login shell and invoke **login**(1). The **.logout** file is not processed. If *username* is omitted, *login* prompts for the name of a user.

**–p** Preserve the current environment (variables).

**logout** Terminate a login shell.

**nice** [*+n* | *–n*] [*command*]

Increment the nice value for the shell or for *command* by *n*. The higher the nice value, the lower the priority of a process, and the slower it runs. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** commands apply. If *command* is omitted, **nice** increments the value for the current shell. If no increment is specified, **nice** sets the nice value to 4. The range of nice values is from –20 to 20. Values of *n* outside this range set the value to the lower, or to the higher boundary, respectively.

*+n* Increment the nice value by *n*.
*–n* Decrement by *n*. This argument can be used only by the super-user.

**nohup** [*command*]

Run *command* with HUPs ignored. With no arguments, ignore HUPs throughout the remainder of a script. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple **if** commands apply. All processes detached with & are effectively **nohup**'d.

**notify** [*%job*] ...

Notify the user asynchronously when the status of the current, or of specified jobs, changes.

**onintr** [ – | *label*]

Control the action of the shell on interrupts. With no arguments, **onintr** restores the default action of the shell on interrupts. (The shell terminates shell scripts and returns to the terminal

command input level). With the – argument, the shell ignores all interrupts. With a *label* argument, the shell executes a **goto** *label* when an interrupt is received or a child process terminates because it was interrupted.

**popd** [+*n*]
Pop the directory stack, and **cd**s to the new top directory. The elements of the directory stack are numbered from 0 starting at the top.

+*n* Discard the *n*'th entry in the stack.

**pushd** [+*n* | *dir*]
Push a directory onto the directory stack. With no arguments, exchange the top two elements.

+*n* Rotate the *n*'th entry to the top of the stack and **cd** to it.
*dir* Push the current working directory onto the stack and change to *dir*.

**rehash** Recompute the internal hash table of the contents of directories listed in the *path* variable to account for new commands added.

**repeat** *count command*
Repeat *command count* times *command* is subject to the same restrictions as with the one-line **if** statement.

**set** [*var* [ = *value* ] ]
**set** *var*[*n*] = *word*
With no arguments, **set** displays the values of all shell variables. Multiword values are displayed as a parenthesized list. With the *var* argument alone, **set** assigns an empty (null) value to the variable *var*. With arguments of the form *var* = *value* **set** assigns *value* to *var*, where *value* is one of:

*word* A single word (or quoted string).
(*wordlist*) A space-separated list of words enclosed in parentheses.

Values are command and filename expanded before being assigned. The form **set** *var*[*n*] = *word* replaces the *n*'th word in a multiword value with *word*.

**setenv** [*var* [ *word*] ]
With no arguments, **setenv** displays all environment variables. With the *var* argument sets the environment variable *var* to have an empty (null) value. (By convention, environment variables are normally given upper-case names.) With both *var* and *word* arguments **setenv** sets the environment variable name to the value *word*, which must be either a single word or a quoted string. The most commonly used environment variables, **USER**, **TERM**, and **PATH**, are automatically imported to and exported from the **csh** variables **user**, **term**, and **path**; there is no need to use **setenv** for these. In addition, the shell sets the PWD environment variable from the **csh** variable *cwd* whenever the latter changes.

**shift** [*variable*]
The components of **argv**, or *variable*, if supplied, are shifted to the left, discarding the first component. It is an error for the variable not to be set, or to have a null value.

**source** [–**h**] *name*
Reads commands from *name*. **source** commands may be nested, but if they are nested too deeply the shell may run out of file descriptors. An error in a sourced file at any level terminates all nested **source** commands.

–**h** Place commands from the the file *name* on the history list without executing them.

**stop** [*%job*] ...
Stop the current or specified background job.

**suspend** Stop the shell in its tracks, much as if it had been sent a stop signal with ^**Z**. This is most often used to stop shells started by **su**.

**switch** (*string*)
**case** *label*:
...
**breaksw**
...
**default:**
...
**breaksw**
**endsw**

Each *label* is successively matched, against the specified *string*, which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a "default" label is found, execution begins after the default label. Each **case** statement and the **default** statement must appear at the beginning of a line. The command **breaksw** continues execution after the **endsw**. Otherwise control falls through subsequent **case** and **default** statements as with C. If no label matches and there is no default, execution continues after the **endsw**.

**time** [*command*]

With no argument, print a summary of time used by this C shell and its children. With an optional *command*, execute *command* and print a summary of the time it uses.

**umask** [*value*]

Display the file creation mask. With *value* set the file creation mask. *value* is given in octal, and is XORed with the permissions of 666 for files and 777 for directories to arrive at the permissions for new files. Common values include 002, giving complete access to the group, and read (and directory search) access to others, or 022, giving read (and directory search) but not write permission to the group and others.

**unalias** *pattern*

Discard aliases that match (filename substitution) *pattern*. All aliases are removed by **unalias** *.

**unhash** Disable the internal hash table.

**unlimit** [**–h**] [*resource*]

Remove a limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed. See the description of the **limit** command for the list of *resource* names.

**–h** Remove corresponding hard limits. Only the super-user may do this.

**unset** *pattern*

Remove variables whose names match (filename substitution) *pattern*. All variables are removed by '**unset** *'; this has noticeably distasteful side-effects.

**unsetenv** *variable*

Remove *variable* from the environment. Pattern matching, as with **unset** is not performed.

**wait** Wait for background jobs to finish (or for an interrupt) before prompting.

**while** (*expr*)
...
**end**

While *expr* is true (evaluates to non-zero), repeat commands between the **while** and the matching **end** statement. **break** and **continue** may be used to terminate or continue the loop prematurely. The **while** and **end** must appear alone on their input lines. If the shell's input is a terminal, it prompts for commands with a question-mark until the **end** command is entered and then performs the commands in the loop.

%[*job*] [&]

Bring the current or indicated *job* to the foreground. With the ampersand, continue running *job* in the background.

@ [*var* =*expr* ]
@ [*var*[*n*] =*expr*

With no arguments, display the values for all shell variables. With arguments, the variable *var*, or the *n*'th word in the value of *var*, to the value that *expr* evaluates to. (If [*n*] is supplied, both *var* and its *n*'th component must already exist.)

If the expression contains the characters >, <, & or |, then at least this part of *expr* must be placed within parentheses.

The operators *=, +=, etc., are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* that would otherwise be single words.

Special postfix operators, + + and – – increment or decrement *name*, respectively.

**Environment Variables and Predefined Shell Variables**

Unlike the standard shell, the C shell maintains a distinction between environment variables, which are automatically exported to processes it invokes, and shell variables, which are not. Both types of variables are treated similarly under variable substitution. The shell sets the variables **argv, cwd, home, path, prompt, shell,** and **status** upon initialization. The shell copies the environment variable **USER** into the shell variable *user*, **TERM** into *term*, and **HOME** into *home*, and copies each back into the respective environment variable whenever the shell variables are reset. **PATH** and **path** are similarly handled. You need only set **path** once in the **.cshrc** or **.login** file. The environment variable **PWD** is set from **cwd** whenever the latter changes. The following shell variables have predefined meanings:

**argv** Argument list. Contains the list of command line arguments supplied to the current invocation of the shell. This variable determines the value of the positional parameters **$1, $2,** and so on.

**cdpath** Contains a list of directories to be searched by the **cd, chdir,** and **popd** commands, if the directory argument each accepts is not a subdirectory of the current directory.

**cwd** The full pathname of the current directory.

**echo** Echo commands (after substitutions), just before execution.

**fignore** A list of filename suffixes to ignore when attempting filename completion. Typically the single word **'.o'**.

**filec** Enable filename completion, in which case the EOT character (^**D**) and the ESC character have special significance when typed in at the end of a terminal input line:

EOT Print a list of all filenames that start with the preceding string.
ESC Replace the preceding string with the longest unambiguous extension.

**hardpaths** If set, pathnames in the directory stack are resolved to contain no symbolic-link components.

**histchars** A two-character string. The first character replaces **!** as the history-substitution character. The second replaces the carat (^) for quick substitutions.

**history** The number of lines saved in the history list. A very large number may use up all of the C shell's memory. If not set, the C shell saves only the most recent command.

**home** The user's home directory. The filename expansion of ~ refers to the value of this variable.

**ignoreeof** If set, the shell ignores EOF from terminals. This protects against accidentally killing a C shell by typing a ^**D**.

**mail** A list of files where the C shell checks for mail. If the first word of the value is a number, it specifies a mail checking interval in seconds (default 5 minutes).

**nobeep** Suppress the bell during command completion when asking the C shell to extend an ambiguous filename.

**noclobber** Restrict output redirection so that existing files are not destroyed by accident. > redirections can only be made to new files. >> redirections can only be made to existing files.

**noglob** Inhibit filename substitution. This is most useful in shell scripts once filenames (if any) are obtained and no further expansion is desired.

**nonomatch**
Returns the filename substitution pattern, rather than an error, if the pattern is not matched. Malformed patterns still result in errors.

**notify** If set, the shell notifies you immediately as jobs are completed, rather than waiting until just before issuing a prompt.

**path** The list of directories in which to search for commands. **path** is initialized from the environment variable **PATH**, which the C shell updates whenever **path** changes. A null word specifies the current directory. The default is typically: **(. /usr/ucb /usr/bin)**. If **path** becomes unset only full pathnames will execute. An interactive C shell will normally hash the contents of the directories listed after reading **.cshrc**, and whenever **path** is reset. If new commands are added, use the **rehash** command to update the table.

**prompt** The string an interactive C shell prompts with. Noninteractive shells leave the **prompt** variable unset. Aliases and other commands in the **.cshrc** file that are only useful interactively, can be placed after the following test: **'if ($?prompt == 0) exit'**, to reduce startup time for noninteractive shells. A **!** in the **prompt** string is replaced by the current event number. The default prompt is *hostname***%** for mere mortals, or *hostname***#** for the super-user.

**savehist** The number of lines from the history list that are saved in **~/.history** when the user logs out. Large values for **savehist** slow down the C shell during startup.

**shell** The file in which the C shell resides. This is used in forking shells to interpret files that have execute bits set, but that are not executable by the system.

**status** The status returned by the most recent command. If that command terminated abnormally, 0200 is added to the status. Built-in commands that fail return exit status 1, all other built-in commands set status to 0.

**time** Control automatic timing of commands. Can be supplied with one or two values. The first is the reporting threshold in CPU seconds. The second is a string of tags and text indicating which resources to report on. A tag is a percent sign (**%**) followed by a single *upper-case* letter (unrecognized tags print as text):

- **%D** Average amount of unshared data space used in Kilobytes.
- **%E** Elapsed (wallclock) time for the command.
- **%F** Page faults.
- **%I** Number of block input operations.
- **%K** Average amount of unshared stack space used in Kilobytes.
- **%M** Maximum real memory used during execution of the process.
- **%O** Number of block output operations.
- **%P** Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time.
- **%S** Number of seconds of CPU time consumed by the kernel on behalf of the user's process.
- **%U** Number of seconds of CPU time devoted to the user's process.
- **%W** Number of swaps.
- **%X** Average amount of shared memory used in Kilobytes.

The default summary display outputs from the **%U**, **%S**, **%E**, **%P**, **%X**, **%D**, **%I**, **%O**, **%F** and **%W** tags, in that order.

**verbose** Display each command after history substitution takes place.

**Sun386i Environment Variables**

**DOS_PRINTER** The value of this environment variable indicates the timeout (in seconds) for printing in a DOS window. A value of **20** (the default) indicates that jobs will be sent to the print spooler after 20 seconds of no printing activity from DOS to that printer. A value of **0** indicates that the spooler must be flushed manually from the menu in the DOS window.

**DOSLOOKUP** If set, this environment variable indicates that a command should be tried as a DOS command if not recognized by SunOS. If the command is supported by DOS, a DOS window will be created and the command executed in that window. Otherwise, the standard "command not found" error message results.

## DIAGNOSTICS

**You have stopped jobs.**

You attempted to exit the C shell with stopped jobs under job control. An immediate second attempt to exit will succeed, terminating the stopped jobs.

## FILES

| | |
|---|---|
| **~/.cshrc** | Read at beginning of execution by each shell. |
| **~/.login** | Read by login shells after **.cshrc** at login. |
| **~/.logout** | Read by login shells at logout. |
| **~/.history** | Saved history for use at next login. |
| **/usr/bin/sh** | Standard shell, for shell scripts not starting with a '#'. |
| **/tmp/sh*** | Temporary file for '<<'. |
| **/etc/passwd** | Source of home directories for '*~name*'. |

## SEE ALSO

**login**(1), **printenv**(1), **sh**(1), **tset**(1), **access**(2), **execve**(2), **fork**(2), **pipe**(2), **termio**(4), **a.out**(5), **environ**(5V), **ascii**(7)

*Doing More with SunOS: Beginner's Guide*
*Getting Started with SunOS: Beginner's Guide*

## LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of **alias** substitutions on a single line to 20.

## BUGS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (that is, wrong) as the job may have changed directories internally.

Shell built-in functions are not stoppable/restartable. Command sequences of the form *a* ; *b* ; *c* are also not handled gracefully when stopping is attempted. If you suspend *b*, the shell never executes *c*. This is especially noticeable if the expansion results from an alias. It can be avoided by placing the sequence in parentheses to force it into a subshell.

Control over terminal output after processes are started is primitive; use the Sun Window system if you need better output control.

Multiline shell procedures should be provided, as they are with the standard (Bourne) shell.

Commands within loops, prompted for by **?**, are not placed in the *history* list.

Control structures should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. There are two problems with : modifier usage on variable substitutions: not all of the modifiers are available, and only one modifier per substitution is allowed.

The **g** (global) flag in history substitutions applies only to the first match in each word, rather than all matches in all words. The the standard text editors consistently do the latter when given the **g** flag in a substitution command.

Quoting conventions are contradictory and confusing.

Symbolic links can fool the shell. Setting the **hardpaths** variable alleviates this.

'**set path**' should remove duplicate pathnames from the pathname list. These often occur because a shell script or a **.cshrc** file does something like '**set path=(/usr/local /usr/hosts $path)**' to ensure that the named directories are in the pathname list.

The only way to direct the standard output and standard error separately is by invoking a subshell, as follows:

**example%** (*command* > *outfile*) >& *errorfile*

Although robust enough for general use, adventures into the esoteric periphery of the C shell may reveal unexpected quirks.

NAME

csplit – split a file with respect to a given context

SYNOPSIS

**csplit** [ **–f** *prefix* ] [ **–k** ] [ **–s** ] *filename argument1* [ *...argumentn* ]

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**csplit** reads the file whose name is *filename* and separates it into *n*+1 sections, defined by the arguments *argument1* through *argumentn*. If the *filename* argument is a '–', the standard input is used. By default the sections are placed in files named **xx00** through **xx***n*. *n* may not be greater than 99. These sections receive the following portions of the file:

| | |
|---|---|
| **xx00** | From the start of *filename* up to (but not including) the line indicated by *argument1* (see OPTIONS below for an explanation of these arguments.) |
| **xx01**: | From the line indicated by *argument1* up to the line indicated by *argument2*. |
| **xx***n*: | From the line referenced by *argumentn* to the end of *filename*. |

**csplit** prints the character counts for each file created, and removes any files it creates if an error occurs.

OPTIONS

| | |
|---|---|
| **–f** *prefix* | name the created files *prefix***00** through *prefixn*. |
| **–k** | Suppress removal of created files when an error occurs. |
| **–s** | Suppress printing of character counts. |

The arguments *argument1* through *argumentn* can be a combination of the following selection operators:

| | |
|---|---|
| */rexp/* | A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line then becomes the line containing *rexp*. This argument may be followed by an optional '+' or '–' some number of lines (for example, **/Page/–5**). |
| *%rexp%* | This argument is the same as */rexp/*, except that no file is created for the selected section. |
| *lineno* | A file is to be created from the current line up to (but not including) *lineno*. The current line becomes *lineno*. |
| {*num*} | Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lineno*, the file will be split every *lineno* lines (*num* times) from that point. |

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines.

EXAMPLES

This example splits the file at every 100 lines, up to 10,000 lines.

```
csplit –k file 100 {99}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

```
csplit –k prog.c '%main(%' '/^}/+1' {20}
```

SEE ALSO

**ed**(1), **sh**(1), **regexp**(3)

**DIAGNOSTICS**

Self-explanatory except for:

**arg – out of range**

which means that the given argument did not refer to a line between the current position and the EOF.

NAME

ctags – create a tags file for use with vi

SYNOPSIS

**ctags** [ **–aBFtuvwx** ] [ **–f** *tagsfile* ] *filename*...

DESCRIPTION

**ctags** makes a tags file for **ex**(1) from the specified C, Pascal, FORTRAN, YACC, and LEX sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by SPACE or TAB characters. Using the tags file, **ex** can quickly find these objects definitions.

Normally **ctags** places the tag descriptions in a file called **tags**; this may be overridden with the **–f** option.

Files with names ending in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Files with names ending in **.y** are assumed to be YACC source files. Files with names ending in **.l** are assumed to be LEX files. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again looking for C definitions.

The tag *main* is treated specially in C programs. The tag formed is created by prepending **M** to *filename*, with a trailing **.c** removed, if any, and leading pathname components also removed. This makes use of **ctags** practical in directories with more than one program.

OPTIONS

**–a** Append output to an existing tags file.

**–B** Use backward searching patterns (**?**...**?**).

**–F** Use forward searching patterns (/.../) (default).

**–t** Create tags for typedefs.

**–u** Update the specified files in tags, that is, all references to them are deleted, and the new values are appended to the file. Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the **tags** file.

**–v** Produce on the standard output an index of the form expected by **vgrind**(1). This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through '**sort –f**'. See EXAMPLES.

**–w** Suppress warning diagnostics.

**–x** Produce a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

EXAMPLES

Using **ctags** with the **–v** option, the output will be sorted into lexicographic order. You may want to run the output through '**sort –f**'.

**ctags –v** *filenames* **| sort –f > index**
**vgrind –x index**

FILES

**tags** output tags file

SEE ALSO

**ex**(1), **vgrind**(1), **vi**(1)

BUGS

Recognition of **functions, subroutines** and **procedures** for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

**ctags** does not know about **#ifdefs.**

**ctags** should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **–tx** shows only the last line of typedefs.

NAME

ctrace – generate a C program execution trace

SYNOPSIS

**ctrace** [ **–f** *functions* ] [ **–v** *functions* ] [ **–o x u e** ] [ **–s P b** ] [ **–l** *n* ] [ **–t** *n* ] [ ] [ **–b** ] [ **–p** '*s*' ] [ **–r** *f* ] [ *filename* ]

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**ctrace** allows you to follow the execution of a C program, statement by statement. The effect is similar to executing a shell procedure with the **–x** option. **ctrace** reads the C program in *filename* (or from standard input if you do not specify *filename*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of **ctrace** into a temporary file because the cc(1V) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed.

A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the **tail**(1) command.

OPTIONS

**–f** *functions* Trace only these *functions*.

**–v** *functions* Trace all but these *functions*.

You may want to add to the default formats for printing variables. **long** and pointer variables are always printed as signed integers. Pointers to character arrays are printed as strings if appropriate. **char**, **short**, and **int** variables are printed as signed integers and, if appropriate, as characters. **double** variables are printed as floating-point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:

**–o** Octal.

**–x** Hexadecimal.

**–u** Unsigned.

**–e** Floating point.

These options are used only in special circumstances:

**–l** *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

**–s** Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

**–t** *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The DIAGNOSTICS section explains when to use this option.

**–P** Run the C preprocessor on the input before tracing it. You can also use the **–D**, **–I**, and **–U** cc(1V) options.

These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX system environment:

**–b** Use only basic functions in the trace code, that is, those in **ctype**(3), **printf**(3S), and **string**(3). These are often available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have **signal**(3), **fflush**, **longjmp** or **setjmp**(3) (see **fclose**(3S) and **setjmp**(3)).

**–p** *'s'* Change the trace print function from the default of '**printf**('. For example, '**fprintf**(**stderr**' would send the trace to the standard error output.

**–r***f* Use file *f* in place of the **runtime.c** trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **–p** option).

## USAGE

### Execution-Time Trace Control

The default operation for **ctrace** is to trace the entire program file, unless you use the **–f** or **–v** options to trace specific functions. This does not give you statement by statement control of the tracing, nor does it let you turn the tracing off and on when executing the tracedprogram.

You can do both of these by adding **ctroff**() and **ctron**() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with **if** statements, and you can even conditionally include this code because **ctrace** defines the CTRACE preprocessor variable. For example:

```
#ifdef
CTRACE
        if (c == '!' && i > 1000)
                ctron();
                #endif
```

You can also call these functions from **dbx**(1) if you compile with the **–g** option. For example, to trace all but lines 7 to 10 in the primary source file, enter:

```
dbx a.out
when at 7 { call ctroff(); cont; }
when at 11 { call ctron(); cont; }
run
```

You can also turn the trace off and on by setting the static variable.B tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly, such as **adb**(1).

## EXAMPLE

If the file **lc.c** contains this C program:

```
#include <stdio.h>
main()  /* count lines in input */
{
        int c, nl;
        nl = 0;
        while ((c = getchar()) != EOF)
                if (c = '\n')
                        ++nl;
        printf("%d\n", nl);
}
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
CTRL-D,
```

the program will be compiled and executed. The output of the program will be the number **2**, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke **ctrace** with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
main()
        nl = 0;
        /* nl == 0 */
        while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
        /* c == 49 or '1' */
                if (c = '\n')
                /* c == 10 or '\n' */
                        ++nl;
                        /* nl == 1 */
        while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
                if (c = '\n')
                /* c == 10 or '\n' */
                        ++nl;
                        /* nl == 2 */
    /* repeating */
```

If you now enter an end of file character (CTRL-D) the final output will be:

```
    /* repeated <1 time */
        while ((c = getchar()) != EOF)
        /* c == -1 */
        printf("%d\n", nl);
        /* nl == 2 */2
    /* return */
```

Program output is printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by **ctrace** at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator = in place of the equal operator ==. You can easily miss this error during code reading.

## FILES

**/usr/lib/ctrace/runtime.c**
run-time trace package

## SEE ALSO

**adb**(1), **cc**(1V), **dbx**(1), **tail**(1), **ctype**(3), **fclose**(3S), **printf**(3S), **setjmp**(3), **signal**(3), **string**(3)

## DIAGNOSTICS

This section contains diagnostic messages from both **ctrace** and **cc**(1V), since the traced code often gets some **cc** warning messages. You can get **cc** error messages in some rare cases, all of which can be avoided.

From **ctrace**

**warning: some variables are not traced in this statement**
Only 10 variables are traced in a statement to prevent the **C** compiler '**out of tree space; simplify expression**' error. Use the **–t** option to increase this number.

**warning: statement too long to trace**
This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

**cannot handle preprocessor code, use –P option**
This is usually caused by **#ifdef/#endif** preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

**' if ... else if ' sequence too long**
Split the sequence by removing an **else** from the middle.

**possible syntax error, try –P option**
Use the **–P** option to preprocess the **ctrace** input, along with any appropriate **–D**, **–I**, and **–U** preprocessor options. If you still get the error message, check the WARNINGS section below.

From cc

**warning: floating-point not implemented**
**warning: illegal combination of pointer and integer**
**warning: statement not reached**
**warning: sizeof returns 0**
Ignore these messages.

**compiler takes size of function**
See the **ctrace 'possible syntax error'** message above.

**yacc stack overflow**
See the **ctrace ''if ... else if' sequence too long'** message above.

**out of tree space; simplify expression**
Use the **–t** option to reduce the number of traced variables per statement from the default of 10. Ignore the **'ctrace: too many variables to trace'** warnings you will now get.

**redeclaration of signal**
Either correct this declaration of **signal**(3), or remove it and **#include <signal.h>**.

Warnings

You will get a **ctrace** syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

**ctrace** assumes that BADMAG is a preprocessor macro, and that EOF and NULL are **#defined** constants. Declaring any of these to be variables, for example, **'int EOF;'**, will cause a syntax error.

BUGS

**ctrace** does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. **ctrace** may choose to print the address of an aggregate or use the wrong format (for example, **%e** for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

NAME

cut – remove selected fields from each line of a file

SYNOPSIS

**cut** **–c** *list* [ *filename* ... ]

**cut** **–f** *list* [ **–d***c* ] [ **–s** ] [ *filename* ... ]

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

Use **cut** to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be of fixed length, such character positions (such as on a punched card), or of variable length between lines. They can be marked with a field delimiter character, such as TAB (as specified with the **–f** option). **cut** can be used as a filter; if no files are given, the standard input is used. In addition, a file name of '–' explicitly refers to standard input.

OPTIONS

**–c***list* By character position. *list* is a comma-separated list of integer field numbers (in increasing order), with an optional '–' to indicate ranges:

**1,4,7** characters 1, 4 and 7
**1–3,8** characters 1 through 3, and 8
**–5,10** characters (1) through 5, and 10
**3–** characters 3 through (last)

**–f** *list* By field position. Instead of character positions, *list* specifies fields that are separated a delimiter (normally a TAB):

**1,4,7** fields 1, 4 and 7

Lines with no field delimiters are normally passed through intact (to allow for subheadings).

**–d***c* Set the field delimiter to *c*. The default is a TAB, SPACE, or a character with special meaning to the shell must be quoted.

**–s** Suppress lines with no delimiter characters.

EXAMPLES

**cut –d: –f1,5 /etc/passwd**
Mapping of user IDs to names.

**name=who am i | cut –f1 –d" "**
Set **name** to the current login name.

SEE ALSO

**grep**(1V), **paste**(1)

DIAGNOSTICS

**ERROR: line too long**
A line can have no more than 1023 characters or fields.

**ERROR: bad list for c/f option**
Missing **–c** or **–f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

**ERROR: no fields**
The *list* is empty.

**ERROR: no delimiter**
Missing *char* on **–d** option.

**ERROR: cannot handle multiple adjacent backspaces**
Adjacent backspaces cannot be processed correctly.

**WARNING: cannot open** *<filename>*: *<reason>*
Either *filename* cannot be read or does not exist. If multiple filenames are present, processing continues.

**WARNING: I/O error reading** *<filename>*: *<reason>*
An I/O error occurred when reading *filename*. If multiple filenames are present, processing continues.

NAME

cxref – generate a C program cross-reference

SYNOPSIS

**cxref** [ **–c** ] [ **–w** [ *num* ] ] [ **–o***filename* ] [ **–t** ] *filenames*

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**cxref** analyzes a collection of C files and attempts to build a cross-reference table. **cxref** utilizes a special option of **cpp**(1) to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **–c** option, in combination. Each symbol contains an asterisk (*****) before the declaring reference.

SYSTEM V DESCRIPTION

The System V version of **cxref**, run as **/usr/5bin/cxref**, makes the C preprocessor search for include files in **/usr/5include** before searching for them in **/usr/include**.

OPTIONS

In addition to the **–D**, **–I** and **–U** options (which are identical to their interpretation by **cc**(1V)), the following options are interpreted by **cxref**:

**–c** Print a combined cross-reference of all input files.

**–w**[ *num* ] Width option which formats output no wider than *num* (decimal) columns. This option will default to 80 if *num* is not specified or is less than 51.

**–o***filename* Direct output to named *file*.

**–s** Operate silently; does not print input file names.

**–t** Format listing for 80-column width.

FILES

**/usr/tmp/xr***

temporary files

SEE ALSO

**cc**(1V), **cpp**(1)

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

BUGS

**cxref** considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that **#includes ctype. h**, will contain many declarations of the variable **c**.

NAME

date – display or set the date

SYNOPSIS

**date** [ **–u** ] [ **–a** [ – ] *sss.fff* ] [ *yy mm dd hh mm* [ *.ss* ] ] [ *+format* ]

SYSTEM V SYNOPSIS

**date** [ **–u** ] [ **–a** [ – ] *sss*.**fff** ] [ *mm dd hh mm* [ *yy* ] ] [ *+format* ]

DESCRIPTION

If no argument is given, or if the argument begins with **+**, **date** displays the current date and time. Otherwise, the current date is set. Only the super-user may **set** the date.

*yy* is the last two digits of the year; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *.ss* (optional) specifies seconds. The year, month and day may be omitted; the current values are supplied as defaults.

If the argument begins with **+**, the output of **date** is under the control of the user. The format for the output is similar to that of the first argument to **printf**(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by **%** and will be replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

| | |
|---|---|
| **n** | insert a new-line character |
| **t** | insert a tab character |
| **m** | month of year – 01 to 12 |
| **d** | day of month – 01 to 31 |
| **y** | last 2 digits of year – 00 to 99 |
| **D** | date as mm/dd/yy |
| **H** | hour – 00 to 23 |
| **M** | minute – 00 to 59 |
| **S** | second – 00 to 59 |
| **T** | time as HH:MM:SS |
| **j** | day of year – 001 to 366 |
| **w** | day of week – Sunday = 0 |
| **a** | abbreviated weekday – Sun to Sat |
| **h** | abbreviated month – Jan to Dec |
| **r** | time in AM/PM notation |

SYSTEM V SYNOPSIS

When setting the date, the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. The current year is the default if no year is mentioned.

OPTIONS

**–u** Display the date in GMT (universal time). The system operates in GMT; **date** normally takes care of the conversion to and from local standard and daylight time. **–u** may also be used to set GMT time.

**–a** [–]*sss.fff* Using the **adjtime**(2) system call, tell the system to slowly adjust the time by *sss.fff* seconds (*fff* represents fractions of a second). This adjustment can be positive or negative. The system's clock will be sped up or slowed down until it has drifted by the number of seconds specified.

EXAMPLES

**date 10080045**

Would set the date to Oct 8, 12:45 AM.

If the year were 1986, and the date were so set,

**date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'**

would generate as output:

**DATE: 08/01/86**
**TIME: 14:45:05**

FILES

| | |
|---|---|
| **/var/adm/wtmp** | to record time-setting |
| **/usr/share/lib/zoneinfo** | timezone definitions |

SEE ALSO

**adjtime**(2), **printf**(3S), **utmp**(5)

DIAGNOSTICS

**date: Failed to set date: Not**
If you try to change the date but are not the super-user.

**date: bad format character**
If the field descriptor is not recognizable.

## NAME

dbx – source-level debugger

## SYNOPSIS

**dbx** [ **–f** *fcount* ] [ **–i** ] [ **–I** *dir* ] [ **–k** ] [ **–kbd** ] [ **–P** *fd* ] [ **–r** ] [ **–s** *startup* ] [ **–sr** *tstartup* ]
[ *objfile* [ *corefile* | *process-id* ] ]

## AVAILABILITY

This command is available with the *Debugging* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**dbx** is a utility for source-level debugging and execution of programs written in C, or other supported languages such as Pascal and FORTRAN 77. **dbx** accepts the same commands as **dbxtool**(1), but uses a standard terminal (tty) interface.

*objfile* is an object file, produced by cc(1V) or another compiler, with the **–g** option to include a symbol table. This symbol table contains the names of all the source files used to create it, and these files are available for perusal while using the debugger.

If no *objfile* is specified, you can use **dbx**'s **debug** command to specify the program to debug.

If there is a file named **core** in the current directory, or a *corefile* argument is specified, you can use **dbx** to examine the state of the program when the core file was produced.

**dbx** commands in the file **.dbxinit** are executed immediately after the symbol table is read, if that file exists in the current directory, or in the user's home directory if **.dbxinit** doesn't exist in the current directory.

## OPTIONS

**–f** *fcount* — Alter the initial estimate of the number functions in the program being debugged. The initial setting is 500.

**–i** — Force **dbx** to act as though the standard input were a terminal or terminal emulator.

**–I** *dir* — Add *dir* to the list of directories in which to search for a source file. **dbx** normally searches the current directory, and the directory where *objfile* is located. The directory search path can be reset with the **use** command.

**–k** — Kernel debugging.

**–kbd** — Debug a program that sets the keyboard into up-down translation mode. This flag is necessary if a program uses up-down decoding.

**–P** *fd* — Create a pipeline to a **dbxtool**(1) process. *fd* is the file descriptor through which to pipe output to the front-end process. This option is passed automatically to **dbx** by **dbxtool**.

**–r** — Execute *objfile* immediately. Parameters follow the object file name (redirection is handled properly). If the program terminates successfully, **dbx** exits. Otherwise, **dbx** reports the reason for termination and waits for a response. **dbx** reads from the terminal (**/dev/tty**) when **–r** is specified and standard input is a file or pipe.

**–s** *startup* — Read initialization commands from the file named *startup*.

**–sr** *tstartup* — Read initialization commands from the temporary file named *startup*, and then remove that file.

## USAGE

Refer to **dbx** in *Debugging Tools*

The most useful basic commands to know about are **run**, to run the program being debugged, **where**, to obtain a stack trace with line numbers, **print**, to display variables, and **stop**, to set breakpoints.

**Filenames**

Filenames in **dbx** may include shell metacharacters. The shell used for pattern matching is determined by the SHELL environment variable.

**Expressions**

**dbx expressions** are combinations of variables, constants, procedure calls, and operators. Variables are either variables in the program being debugged or special **dbx** variables whose names begin with $. Hexadecimal constants must be preceded by a '0x' and octal constants by a '0'. Character constants must be enclosed in single quotes. Expressions cannot involve strings, structures, or arrays, although elements of structures or arrays may be used.

**Operators**

| | |
|---|---|
| + − * / **div** % | Add, subtract, multiply, divide, integer division, and remainder, respectively. |
| << >> & \| | Left-shift, right-shift, bitwise AND, bitwise OR, and bitwise complement. |
| & * | Address of operator, and contents of operator. |
| < > <= >= == != ! | Less than, greater than, less than or equal to, greater than or equal to, equal, not equal, and negation. |
| && \|\| | Logical AND, and logical OR |
| **sizeof** (*cast*) | Size of variable or type and type cast. |
| . −> | Field reference, and pointed-to-field reference (however, dot works for both in **dbx**). |

Precedence and associativity of operators are the same as for C; parentheses can be used for grouping.

If there is no *corefile*, only expressions containing constants are available. Procedure calls require an active child process.

**Scope Rules**

**dbx** uses the current file and function to resolve scope conflicts. Their values are updated as files and functions are entered and exited during execution. You can also change them explicitly by using the **file** and **func** commands. When the current function is changed, the current file is updated along with it, but not vice versa.

**Execution and Tracing Commands**

^C Interrupt. Stop the program being debugged and enter **dbx**.

**run** [ *args* ] [ < *infile* ] [ >| >> *outfile* ]

Start executing *objfile*, reading in any new information from it. With no *args*, use the argument list from the previous **run** command.

| | |
|---|---|
| *args* | Pass *args* as command-line arguments to the program. |
| <\|>\|>> | Redirect input or output, or append output to a file. |

**rerun** [ *args* ] [ < *infile* ] [ >| >> *outfile* ]

Like the **run** command, except that when *args* are omitted, none are passed to the program.

**cont** [ **at** *sourceline* ] [ **sig** *signal* ]

Continue execution from where it stopped.

| | |
|---|---|
| **at** *sourceline* | Start from *sourceline* |
| **sig** *signal* | Continue as if *signal* had occurred. *signal* may be a number or a name as with **catch**. |

**trace** [ **in** *function* ] [ **if** *condition* ]
**trace** *sourceline* [ **if** *condition* ]
**trace** *function* [ **if** *condition* ]
**trace** *expression* **at** *sourceline* [ **if** *condition* ]
**trace** *variable* [ **in** *function* ] [ **if** *condition* ]

Display tracing information. If no argument is specified, each source line is displayed before execution. Tracing is turned off when the function or procedure is exited.

| | |
|---|---|
| **in** *function* | Display tracing information only while executing the function or procedure *function*. |
| **if** *condition* | Display tracing information only if *condition* is true. |
| *sourceline* | Display the line immediately prior to executing it. Source line-numbers from another file are written as *filename:***n.** |
| *function* | Display the routine and source line called from, parameters passed in, and return value. |
| *expression* **at** *sourceline* | Display the value of *expression* whenever *sourceline* is reached. |
| *variable* | Display the name and value whenever *variable* changes. |

**stop at** *sourceline* [ **if** *condition* ]
**stop in** *function* [ **if** *condition* ]
**stop** *variable* [ **if** *condition* ]
**stop if** *condition*

Stop execution when the *sourceline* is reached, *function* is called, *variable* is changed, or *condition* becomes true.

**when in** *function* { *command* **;** [ *command* **;** ] ... }
**when at** *sourceline* { *command* **;** [ *command* **;** ] ... }
**when** *condition* { *command* **;** [ *command* **;** ] ... }

Execute the **dbx** *command*(s) when *function* is called, *sourceline* is reached, or *condition* is true.

**status** [ > *filename* ]

Display active **trace**, **stop** and **when** commands, and associated command numbers.

**delete all**
**delete** *cmd-no* [ , *cmd-no* ] ...

Remove all **traces**, **stops** and **whens**, or those corresponding to each **dbx** *cmd-no* (as displayed by **status**).

**clear** [ *sourceline*]

Clear all breakpoints at the current stopping point, or at *sourceline*.

**catch** [ *signal* [ , *signal* ] ... ]

Display all signals currently being caught, or catch *signal* before it is sent to the program being debugged. A signal can be specified either by name (with the SIG prefix omitted, as with **kill**(1)) or number. Initially all signals are caught except SIGHUP, SIGEMT, SIGFPE, SIGKILL, SIGALRM, SIGTSTP, SIGCONT, SIGCHLD, and SIGWINCH.

**ignore** [ *signal* [ , *signal* ] ... ]

Display all signals currently being ignored, or stop catching *signal*, which may be specified by name or number as with **catch**.

**step** [*n* ]

Execute the next *n* source lines. If omitted, *n* is taken to be 1. Steps into functions.

**next** [*n* ]

Execute the next *n* source lines. If omitted, *n* is taken to be 1. **next** steps past functions.

**Naming, Printing and Displaying Data**

Variables from another function or procedure with the same name as one in the current block must be qualified as follows:

[`*sourcefile*`]*function*`*variable*

For Pascal variables there may be more than one *function* or procedure name, each separated by a backquote.

**print** *expression* [ , *expression* ] ...
Print the value of each *expression*, which may involve function calls. Program execution halts when a breakpoint is reached, and **dbx** resumes.

**display** [ *expression* [ , *expression* ] ... ]
Print a list of the expressions currently being displayed, or display the value of each *expression* whenever execution stops.

**undisplay** [ *expression* [ , *expression* ] ... ]
Stop displaying the value of each *expression* whenever execution stops. If *expression* is a constant, it refers to a display-number as shown by the **display** command with no arguments.

**whatis** *identifier*
**whatis** *type*
Print the declaration of the given identifier or type. *type*s are useful to print all the members of a structure, union, or enumerated type.

**which** *identifier*
Print the fully-qualified name of the given identifier.

**whereis** *identifier*
Print the fully qualified name of all symbols matching *identifier*.

**assign** *variable* = *expression*
**set** *variable* = *expression*
Assign the value of *expression* to *variable*. There is no type conversion for operands of differing type.

**set81** *fpreg*=*word1 word2 word3*
Treat the concatenation of *word1 word2 word3* as a 96-bit, IEEE floating-point value and assign it to the MC68881 floating-point register *fpreg*. (Supported only on Sun-3).

**call** *function*(*parameters*)
Execute the named function. Arguments are passed according to the rules for the source-language of *function*.

**where**[ *n*]
List all, or the top *n*, active functions on the stack.

**dump** [ *function* ]
Display the names and values of local variables and parameters in the current or specified *function*.

**up** [*n*]
**down** [*n*]
Move up (towards "main") or down the call stack, one or *n* levels.

**File Access Commands**

**edit** [ *filename* | *function* ]
Edit the current source file, or the given *filename* or the file that contains *function*.

**file** [ *filename* ]
Print the name of the current source file, or change the current source file to *filename*.

**func** [ *function* | *program* | *objfile* ]
Print the name of the current function, or change to the given *function*, *program*, or *objfile*. Also changes the current scope.

**list** [ *startline* [ , *endline* ] ]
**list** *function*
List the next ten lines from current source file, list from *startline* through *endline*, or and list from five lines above, to five lines below the first line of *function*.

**use** [ *directory-list* ]
Print or set the list of directories in which to search for source files.

**cd** [ *directory* ]
Change the current working directory for **dbx** to *directory* (or to the value of the HOME environment variable).

**pwd** Print the current working directory for **dbx**.

*/reg-exp*[/]
*?reg-exp* [?]
Search the current file for the regular expression *reg-exp*, from the next (previous) line to the end (top). The matching line becomes the new current line.

**Miscellaneous Commands**

**sh** *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

**alias** *new-command-name character-sequence*
Respond to *new-command-name* as though it were *character-sequence*. Special characters occurring in *character-sequence* must be enclosed in quotation marks. Alias substitution as with the C shell (**csh**(1)) also occurs.

**help** [ *command* ]
Display a synopsis of **dbx** commands, or print a short message explaining *command*.

**make** Invoke **make**(1) with the name of the program as its argument. Any arguments set using **dbxenv makeargs** are also passed as arguments.

**source** *filename*
Read and execute **dbx** commands from *filename*. Useful when the *filename* has been created by redirecting an earlier **status** command.

**quit** Exit **dbx**.

**dbxenv**
**dbxenv case sensitive | insensitive**
**dbxenv fpaasm on | off**
**dbxenv fpabase a[0-7] | off**
**dbxenv makeargs** *string*
**dbxenv stringlen** *num*
**dbxenv speed** *seconds*
Display **dbx** attributes or set the named attribute:

| | |
|---|---|
| **case** | Controls whether upper- and lower-case characters are treated as different values. The default is **sensitive**. |
| **fpaasm** | Controls FPA instruction disassembly. The default is **on**. (Supported only on Sun-3). |
| **fpabase** | Sets the base register for FPA instruction disassembly. The default is **off**. (Supported only on Sun-3 systems). |
| **makeargs** | Sets arguments to pass to **make**. The default is **CC=cc –g**. |

**speed** Set the interval between execution during tracing. The default is 0.5 seconds.

**stringlen** Controls the maximum number of characters printed for a "char *" variable in a C program. The default is 512.

**debug** [ **−k** ] [ *objfile* [ *corefile* | *pid* ] ]
With no arguments, print the name of the current program. With arguments, stop debugging the current program and begin debugging *objfile* having either *corefile* or the current process ID *pid*. The **−k** options indicates kernel debugging.

**kill** Stop debugging of the current program, but be ready to debug another.

**detach** Detach the current program (process) from **dbx**. **dbx** will be unable to access or modify its state.

**proc** [ *pid* ]
For kernel debugging. Display which process is mapped into the user area, or map *pid* to the user area.

**Machine-Level Commands**

**tracei** [ *address* ] [ **if** *condition* ]
**tracei** [ *variable* ] [**at** *address* ] [ **if** *condition* ]
Trace execution of a specific machine-instruction address.

**stopi** [ *variable* ] [ **if** *condition* ]
**stopi** [**at** *address* ] [ **if** *condition* ]
Set a breakpoint at a machine instruction address.

**stepi**
**nexti** Single step as in **step** or **next**, but do a single machine instruction rather than a source line.

*address*,*address* / [ *mode* ]
*address* / [*count* ] [ *mode* ]
Display the contents of memory starting at the first (or current) *address* up to the second *address*, or until *count* items have been displayed. The initial display *mode* is **X**. The following modes are supported:

| | |
|---|---|
| **i** | the machine instruction |
| **d** | word in decimal |
| **D** | longword in decimal |
| **o** | word in octal |
| **O** | longword in octal |
| **x** | word in hexadecimal |
| **X** | longword in hexadecimal |
| **b** | byte in octal |
| **c** | byte as a character |
| **s** | strings as characters terminated by a null |
| **f** | single precision real number |
| **F** | double-precision real number |
| **E** | extended-precision real number (not supported on Sun-4) |

An *address* can be specified as an item from the following list, as an expression made up of other addresses and the operators '+', '−', '*', and indirection (unary '*'), or as an arbitrary expression enclosed in parentheses.

| | |
|---|---|
| *&name* | symbolic address |
| *integer* | numeric address |

*address* = [ *mode* ]
Display the value of the *address*.

**Machine Registers**

The machine registers for the current machine type are represented as special **dbx** variables. They can be used in expressions as any other **dbx** variable can. The registers and their variable names are:

*Sun-2 and Sun-3 Registers*

| | |
|---|---|
| **$d[0-7]** | data registers |
| **$a[0-7]** | address registers |
| **$fp** | frame pointer, equivalent to register a6 |
| **$sp** | stack pointer, equivalent to register a7 |
| **$pc** | program counter |
| **$ps** | program status |

*Sun-3-Only Registers*

| | |
|---|---|
| **$fp[0-7]** | MC68881 data registers |
| **$fpc** | MC68881 control register |
| **$fps** | MC68881 status register |
| **$fpi** | MC68881 instruction address register |
| **$fpf** | MC68881 flags register (unused, idle, busy) |
| **$fpa[0-31]** | double-precision interpretation of FPA registers. |
| **$sfpa[0-31]** | single-precision interpretation of FPA registers. |

*Sun-4 Registers*

| | |
|---|---|
| **$g[0-7]** | global registers |
| **$o[0-7]** | "out" registers |
| **$i[0-7]** | "in" registers |
| **$l[0-7]** | "local" registers |
| **$fp** | frame pointer, equivalent to register i6 |
| **$sp** | stack pointer, equivalent to register o6 |
| **$y** | Y register |
| **$psr** | processor state register |
| **$wim** | window invalid mask register |
| **$tbr** | trap base register |
| **$pc** | program counter |
| **$npc** | next program counter |
| **$f[0-31]** | FPU "f" registers |
| **$fsr** | FPU status register |
| **$fq** | FPU queue |

*Sun386i Registers*

| | |
|---|---|
| **$ss** | stack segment register |
| **$eflags** | flags |
| **$cs** | code segment register |
| **$eip** | instruction pointer |
| **$eax** | general register |
| **$ecx** | general register |
| **$edx** | general register |
| **$ebx** | general register |
| **$esp** | stack pointer |
| **$ebp** | frame pointer |
| **$esi** | source index register |
| **$edi** | destination index register |
| **$ds** | data segment register |
| **$es** | alternate data segment register |
| **$fs** | alternate data segment register |
| **$gs** | alternate data segment register |

Registers for the 80386 lower halves (16 bits) are:

| | |
|---|---|
| **$ax** | general register |
| **$cx** | general register |
| **$dx** | general register |
| **$bx** | general register |
| **$sp** | stack pointer |
| **$bp** | frame pointer |
| **$si** | source index register |
| **$di** | destination index register |
| **$ip** | instruction pointer, lower 16 bits |
| **$flags** | flags, lower 16 bits |

The first four Sun386i 16-bit registers can be split into 8-bit parts:

| | |
|---|---|
| **$al** | lower (right) half of register **$ax** |
| **$ah** | higher (left) half of register **$ax** |
| **$cl** | lower (right) half of register **$cx** |
| **$ch** | higher (left) half of register **$cx** |
| **$dl** | lower (right) half of register **$dx** |
| **$dh** | higher (left) half of register **$dx** |
| **$bl** | lower (right) half of register **$bx** |
| **$bh** | higher (left) half of register **$bx** |

Registers for the 80387 are:

| | |
|---|---|
| **$fctrl** | control register |
| **$fstat** | status register |
| **$ftag** | tag register |
| **$fip** | instruction pointer offset |
| **$fcs** | code segment selector |
| **$fopoff** | operand pointer offset |
| **$fopsel** | operand pointer selector |
| **$st0 - $st7** | data registers |

ENVIRONMENT

**dbx** checks the environment variable EDITOR for the name of the text editor to use with the **edit** command.

FILES

| | |
|---|---|
| **core** | default core file |
| **.dbxinit** | local **dbx** initialization file |
| **/.dbxinit** | user's **dbx** initialization file |

SEE ALSO

**cc**(1V), **csh**(1), **dbxtool**(1), **kill**(1), **lex**(1), **make**(1), **yacc**(1)

*Debugging Tools*

BUGS

**dbx** does not correctly handle **C** variables that are local to compound statements. When printing these variables it often gives incorrect results.

**dbx** does not handle FORTRAN entry points well — it treats them as if they were independent routines.

**dbx** does not handle *assigning* to FORTRAN complex types correctly (see the **assign/set** command).

Some operations behave differently in **dbx** than in **C**:

- **dbx** has two division operators — / always yields a floating-point result and **div** always yields an integral result.
- An array or function name does not signify the address of the array or function in **dbx**. An array name signifies the entire array, and a function name signifies a call to the function with no arguments. The address of an array can be obtained by taking the address of its first element, and the address of a function can be obtained by taking the address of its name.

Casts do not work with FORTRAN 77 or Pascal.

Executable code incorporated into a source file using an **#include** preprocessor directive confuses **dbx**.

**dbx** is confused by the output of program generators such as **yacc**(1) and **lex**(1).

You cannot use **dbx** to debug a shared library directly. You can, however, debug a program that *uses* a shared library.

NAME

dbxtool – SunView interface for the dbx source-level debugger

SYNOPSIS

**dbxtool** [ **–d** ] [ **–i** ] [ **–k** ] [ **–kbd** ] [ **–I** *directory* ] [ *objectfile* [ *corefile* ] ]

AVAILABILITY

This command is available with the *Debugging* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**dbxtool**, a source-level debugger for C, Pascal and FORTRAN 77 programs, is a standard tool that runs within the *SunView* environment. It accepts the same commands as **dbx**, but provides a more convenient user interface.

You can use the mouse to set breakpoints, examine the values of variables, control execution, peruse source files, and so on. **dbxtool** has separate subwindows for viewing source code, entering commands and other uses.

*objectfile* is an object file produced by cc(1V), any other Sun compiler, (or a combination of them) with the appropriate flag (**–g**) specified to produce symbol information in the object file. **IMPORTANT:** every stage of the compilation process, including the linking phase, must include the **–g** option. If no *objectfile* is specified, you can use the **debug** command to specify the program to be debugged. The object file contains a symbol table which includes the names of all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named **core** exists in the current directory or a *corefile* is specified, **dbxtool** can be used to examine the state of the program when it faulted.

Debugger commands in the file **.dbxinit** are executed immediately after the symbolic information is read, if that file exists in the current directory, or in the user's home directory if **.dbxinit** does not exist in the current directory.

OPTIONS

**–d** Produce debugging information for the pipeline from which it reads **dbx**(1) output.

**–i** Force **dbxtool** to act as though standard input were a terminal.

**–k** Kernel debugging.

**–kbd** Debugs a program that sets the keyboard into up/down translation mode. This flag is necessary if you are debugging a program that uses up/down encoding.

**–I** *directory*

Add *directory* to the list of directories that are searched when looking for a source file. Normally **dbxtool** looks for source files in the current directory and then in the directory where *objectfile* is located. The directory search path can also be set with the **use** command. Multiple **–I** options may be given.

USAGE

Refer to **dbx**(1) for a summary of **dbx** commands, or *Debugging Tools* for more complete information on using **dbxtool**.

FILES

| | |
|---|---|
| **core** | default core file |
| **.dbxinit** | local **dbx** initialization file |
| **/.dbxinit** | user's **dbx** initialization file |

SEE ALSO

**cc**(1V), **dbx**(1)
*Debugging Tools*
*SunView 1 Programmer's Guide*

## BUGS

The bugs for **dbx**(1) apply to **dbxtool** as well.
The interaction between scrolling in the *source* subwindow and **dbx**'s regular expression search commands is wrong. Scrolling should affect where the next search begins, but it does not.

NAME

dc – desk calculator

SYNOPSIS

**dc** [ *filename* ]

DESCRIPTION

**dc** is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but an input base, output base, and a number of fractional digits to be maintained may be specified. The overall structure of **dc** is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, and then from the standard input.

Note: **bc**(1) is a preprocessor for **dc** that provides infix (normal arithmetic) notation, a C-like syntax for functions, and reasonable control structures for programs.

The following input constructs are recognized:

**Commands**

*number* Push a number onto the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore '_' to input a negative number, and may contain decimal points.

+ − / * % ^
The top two values on the stack are: added (+), subtracted (−), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack and the result is pushed in their place. Any fractional part of an exponent is ignored.

**s***x* Pop the top of the stack and store into a register named *x*, where *x* is any character.

**S***x* Treat *x* as a stack and push the value onto it.

**l***x* Push the value in register *x* onto the stack. The register *x* is not altered. All registers start with zero value.

**L***x* Treat register *x* as a stack, and pop its top value onto the main stack.

**d** Duplicate the top value on the stack.

**p** Print the top value on the stack. The top value remains unchanged. With

**P** Interpret the top of the stack as an ASCII string, remove and print it.

**f** Print all values on the stack and in registers.

**q** Exit the program. If executing a string, pop the recursion level by two.

**Q** Pop the top value on the stack, and pop the string execution level by that value.

**x** Treat the top element of the stack as a character string and execute it as a string of **dc** commands.

**X** Replace the number on the top of the stack with its scale factor.

[...] Put the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x* Pop and compare top two elements of the stack. Execute register *x* if they obey the stated relation.

**v** Replace the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

**!** Interpret the rest of the line as a command.

**c** Clear all values on the stack.

**i** Pop the top value on the stack and use that value as the input radix.

**I** Push the input base on the top of the stack.

**o** Pop the top value on the stack and use as the output radix.

**O** Push the output base on the top of the stack.

**k** The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

**z** Push the stack level onto the stack.

**Z** Replace the number on the top of the stack with its length.

**?** Take a line of input from the input source (usually the terminal) and execute it.

**; :** Used by **bc** for array operations.

EXAMPLE

Print the first ten values of n!

```
[ la1 + dsa * pla10 > y ]sy
0sa1
lyx
```

SEE ALSO

**bc**(1)

DIAGNOSTICS

| | |
|---|---|
| **x is unimplemented** | Where x is an octal number. |
| **stack empty** | For not enough elements on the stack to do what was asked. |
| **Out of space** | When the free list is exhausted (too many digits). |
| **Out of headers** | For too many numbers being kept around. |
| **Out of pushdown** | For too many items on the stack. |
| **Nesting Depth** | For too many levels of nested execution. |

BUGS

Base conversions on fractions are truncated to the number of fractional digits of the input value. The values are not rounded.

NAME

dd – convert and copy files with various data formats

SYNOPSIS

**dd** [ *option*=*value* ] ...

DESCRIPTION

**dd** copies a specified input file to a specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

OPTIONS

**if**=*name* Input file is taken from *name*; standard input is default.

**of**=*name* Output file is taken from *name*; standard output is default. Note: **dd** creates an explicit output file; therefore the **seek** option is usually useless with explicit output except in special cases such as using magnetic tape or raw disk files.

**ibs**=*n* Input block size *n* bytes (default 512).

**obs**=*n* Output block size *n* bytes (default 512).

**bs**=*n* Set both input and output block size, superseding **ibs** and **obs**; also, if no conversion is specified, it is particularly efficient since no copy need be done

**cbs**=*n* Conversion buffer size.

**skip**=*n* Skip *n* input records before starting copy

**files**=*n* Copy *n* input files before terminating (makes sense only when input is a magtape or similar device).

**seek**=*n* Seek *n* records from beginning of output file before copying. This option generally only works with magnetic tapes and raw disk files and is otherwise usually useless if the explicit output file was named with the **of** option.

**count**=*n* Copy only *n* input records.

**conv=ascii** Convert EBCDIC to ASCII.

**ebcdic** Convert ASCII to EBCDIC.

**ibm** Slightly different map of ASCII to EBCDIC.

**block** Convert variable length records to fixed length.

**unblock** Convert fixed length records to variable length.

**lcase** Map alphabetics to lower case.

**ucase** Map alphabetics to upper case.

**swab** Swap every pair of bytes.

**noerror** Do not stop processing on an error.

**sync** Pad every input record to **ibs**.

*arg*, *arg* [,...] Several comma-separated conversions, for a combination of effects. For instance, **conv=sync,block** is useful for reading variable-length output from a pipe.

Where sizes are specified, a number of bytes is expected. A number may end with **k** (kilobytes) to specify multiplication by 1024, **b** (blocks of 512 bytes) to specify multiplication by 512, or **w** (words) to specify multiplication by 4; a pair of numbers may be separated by **x** to indicate a product.

**cbs** is used only if **ascii**, **unblock**, **ebcdic**, **ibm**, or **block** conversion is specified. In the first two cases, **cbs** characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and NEWLINE added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size **cbs**.

After completion, **dd** reports the number of whole and partial input and output blocks.

**EXAMPLE**

To read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x:*

**example% dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase**

Note: the use of raw magtape: **dd** is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

**SEE ALSO**

**cp**(1), **tr**(1V)

**DIAGNOSTICS**

*f+p* **records in(out):**

Numbers of full and partial records read(written).

**BUGS**

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The **ibm** conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

The **block** and **unblock** options cannot be combined with the **ascii, ebcdic** or **ibm.** Invalid combinations silently ignore all but the last mutually-exclusive keyword.

NAME

defaultsedit, defaults_from_input, defaults_to_indentpro, defaults_to_mailrc, indentpro_to_defaults, input_from_defaults, lockscreen_default, mailrc_to_defaults, scrolldefaults, stty_from_defaults – create or edit default settings for SunView 1 and SunView utilities

SYNOPSIS

**defaultsedit** [ *generic-tool-arguments* ]

**defaults_from_input**

**defaults_to_indentpro**

**defaults_to_mailrc**

**indentpro_to_defaults**

**input_from_defaults**

**lockscreen_default**

**mailrc_to_defaults**

**scrolldefaults**

**stty_from_defaults**

AVAILABILITY

These commands are available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**defaultsedit** is a SunView application that provides a convenient means for inspecting and setting default parameters. It can be viewed as an alternative to the traditional UNIX operating system *.*rc* files that contain initialization options for various commands. Currently, you can use **defaultsedit** to manipulate options to the programs **indent**(1), **mail**(1) and **mailtool**(1), **stty**(1V), and **defaultsedit**, as well as the *menu*, *scrollbar*, *text subwindow* and *tty subwindow* packages, and the SunView environment itself.

The remaining commands are used by **defaultsedit** to perform conversions and other functions; they can also be invoked directly from the shell:

| | |
|---|---|
| **defaults_from_input** | update window-system I/O defaults from current system values |
| **defaults_to_indentpro** | update **indent**(1) defaults in the database from the **.indent.pro** file |
| **defaults_to_mailrc** | update the **.mailrc** file from the defaults database |
| **indentpro_to_defaults** | update **indent** defaults from the **.indent.pro** file |
| **input_from_defaults** | update current system values for window-system I/O from defaults database |
| **lockscreen_default** | apply current default for **lockscreen**(1) display program |
| **mailrc_to_defaults** | update **mail**(1) and/or **mailtool**(1) defaults from the **.mailrc** file |
| **scrolldefaults** | a SunView application that lets you try out different settings from the **Scrollbar** category interactively |
| **stty_from_defaults** | set terminal (TTY) options from defaults database |

Any program or package that a user can customize by setting or changing a parameter could be written so as to get its initialization options from the defaults database. For further information, see *SunView 1 System Programmer's Guide*.

## OPTIONS

**defaultsedit** accepts all of the generic tool arguments discussed in **sunview**(1).

## USAGE

This only applies to **defaultsedit**.

### Subwindows

**defaultsedit** consists of four subwindows. From top to bottom they are:

control — Contains the name of the category currently displayed, and buttons labeled SAVE, QUIT, RESET, and EDIT ITEM. To change the category, click the LEFT mouse button on the word CATEGORY, or use the menu that pops up when you click the RIGHT mouse button.

message — A small text subwindow where messages from **defaultsedit** are displayed.

parameters — Shows all current default parameter names with corresponding values. Clicking the LEFT mouse button over a parameter displays a help string in the message subwindow.

edit — A small text subwindow which enables text editing of parameter values. This is useful for very long text values, such as a long mailing list.

### Control Panel

SAVE — Save the current values that differ from the standard defaults in your private database — that is, the **.defaults** file in your home directory.

QUIT — Exit without saving any changes.

RESET — Reset the default parameters of the current category to the values in your private database. This is useful if you change some values, then change your mind and want to restore the original values.

EDIT ITEM — Clicking the RIGHT mouse button over the EDIT ITEM button brings up a menu with three choices: COPY ITEM, DELETE ITEM and EDIT LABEL. Only text or numeric items can be edited. Note: edits made using this menu will appear only in your private defaults database, not in the master database. The three editing operations are described below.

COPY ITEM — Choosing COPY ITEM will duplicate the current item. You can then edit both the label and the value of the newly created item. Only items with text or numeric values can be copied in this way. COPY ITEM is useful when you want to change the number of instances of a certain type of item — for example, to insert a new mail alias into your defaults database.

DELETE ITEM — Choosing DELETE ITEM will delete the current item from your private database. It cannot be permanently deleted if the corresponding node is present in the master database. However, you can make it behave like an undefined node by giving it the special value *255Undefined255*.

EDIT LABEL — Choosing EDIT LABEL allows you to edit the label of the current item. When you choose EDIT LABEL, the label of the current item changes from bold to normal face. Then you can select the label and edit it as a normal panel text item.

Note: SunView starts up faster when you set the **Private_only** parameter in the **Defaults** category to **TRUE**, in which case only your private **.defaults** file is read.

## ENVIRONMENT

DEFAULTS_FILE — The value of this environment variable indicates the file from which private SunView defaults are read. When it is undefined, defaults are read from the **.defaults** file in your home directory.

FILES

**/usr/lib/defaults/*.d** System-wide parameters and their standard settings. Each file is a category in **defaultsedit.**

**/.defaults**

SEE ALSO

**indent**(1), **lockscreen**(1), **mail**(1), **mailtool**(1), **stty**(1V), **sunview**(1)

*SunView 1 Beginner's Guide*

*SunView 1 System Programmer's Guide*

BUGS

Editing of choice items or categories is not supported by **defaultsedit.** Neither is editing of the master defaults database — to add a new program to the master defaults database, you have to edit a master defaults textfile.

**defaultsedit** reorders mail aliases that appear in the **.mailrc** file. This can adversely affect recursive mail aliases. To avoid this, use the **source** command for **mail**(1) to include a file containing such aliases.

NAME

delta – make a delta (change) to an SCCS file

SYNOPSIS

**/usr/sccs/delta** [ **–nps** ] [ **–g** *list* ] [ **–m** [ *mrlist* ] [ **–r***SID* ] ] [ **–y** [ *comment* ] ] *filename* ...

DESCRIPTION

**delta** permanently introduces into the named SCCS file changes that were made to the file retrieved by **get**(1) (called the **g-file**, or generated file).

**delta** makes a delta to each named SCCS file. If a directory is named, **delta** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with 's.') and unreadable files are silently ignored. If a name of '–' is given, the standard input is read (see WARNINGS); each line of the standard input is taken to be the name of an SCCS file to be processed.

**delta** may issue prompts on the standard output depending upon certain options specified and flags (see **admin**(1)) that may be present in the SCCS file (see **–m** and **–y** options below).

OPTIONS

Options apply independently to each named file.

**–n** Retain the edited **g-file** which is normally removed at completion of delta processing.

**–p** Display (on the standard output) the SCCS file differences before and after the delta is applied in a **diff**(1) format.

**–s** Do not display the created delta's SID, number of lines inserted, deleted and unchanged in the SCCS file.

**–g** *list* Specify a *list* of deltas to be *ignored* when the file is accessed at the change level (SID) created by this delta. See **get**(1) for the definition of *list*.

**–m** [ *mrlist* ]

If the SCCS file has the **v** flag set (see **admin**(1)), a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

If **–m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see **–y** option).

MRs in a list are separated by SPACE and/or TAB characters. An unescaped NEWLINE character terminates the MR list.

Note: if the **v** flag has a value (see **admin**(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, **delta** terminates (it is assumed that the MR numbers were not all valid).

**–r***SID* Uniquely identify which delta is to be made to the SCCS file. The use of this option is necessary only if two or more outstanding **get**'s for editing ('**get –e**') on the same SCCS file were done by the same person (login name). The SID value specified with the **–r** option can be either the SID specified on the **get** command line or the SID to be made as reported by the **get** command. A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

**–y** [ *comment* ]

Arbitrary text to describe the reason for making the delta. A null string is considered a valid *comment*.

If **–y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped NEWLINE character terminates the comment text.

FILES

All files of the form *?*-**file** are explained in the **sccs** chapter of *Programming Utilities and Libraries*. The naming convention for these files is also described there.

| | |
|---|---|
| **g.file** | Existed before the execution of **delta**; removed after completion of **delta**. |
| **p.file** | Existed before the execution of **delta**; may exist after completion of **delta**. |
| **q.file** | Created during the execution of **delta**; removed after completion of **delta**. |
| **x.file** | Created during the execution of **delta**; renamed to SCCS file after completion of **delta**. |
| **z.file** | Created during the execution of **delta**; removed during the execution of **delta**. |
| **d.file** | Created during the execution of **delta**; removed after completion of **delta**. |
| **/usr/bin/diff** | Program to compute differences between the "gotten" file and the **g.file**. |

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see **sccsfile**(5)) and produces an error.

A **get** of many SCCS files, followed by a **delta** of those files, should be avoided when the **get** generates a large amount of data. Instead, multiple **get/delta** sequences should be used.

If the standard input ('**-**') is specified on the **delta** command line, the **-m** (if necessary) and **-y** options *must* also be present. Omission of these options is an error.

SEE ALSO

**admin**(1), **diff**(1), **get**(1), **help**(1), **prs**(1), **sccs**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

DIAGNOSTICS

Use **help**(1) for explanations.

**NAME**

deroff – remove nroff, troff, tbl and eqn constructs

**SYNOPSIS**

**deroff** [ **–w** ] *filename* ...

**AVAILABILITY**

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**deroff** reads each file in sequence and removes all **nroff** and *troff* command lines, backslash constructions, macro definitions, *eqn* constructs (between .EQ and .EN lines or between delimiters), and table descriptions and writes the remainder on the standard output. **deroff** follows chains of included files (**.so** and **.nx** commands); if a file has already been included, a **.so** is ignored and a **.nx** terminates execution. If no input file is given, **deroff** reads from the standard input file.

**OPTIONS**

**–w** Generate a word list, one word per line. A 'word' is a string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed. All other characters are ignored.

**SEE ALSO**

**eqn**(1), **nroff**(1), **tbl**(1), **troff**(1)

**BUGS**

**deroff** is not a complete **troff** interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

**deroff** does not work well with files that use **.so** to source in the standard macro package files.

NAME

des – encrypt or decrypt data using Data Encryption Standard

SYNOPSIS

**des** **–e** | **–d** [ **–b** ] [ **–f** ] [ **–k** *key* ] [ **–s** ] [ *infile* [ *outfile* ] ]

DESCRIPTION

**des** encrypts and decrypts data using the NBS Data Encryption Standard algorithm. One of **–e** (for encrypt) or **–d** (for decrypt) must be specified.

The **des** command is provided to promote secure exchange of data in a standard fashion.

OPTIONS

**–b** Select ECB (eight bytes at a time) encryption mode.

**–f** Suppress warning message when software implementation is used.

**–k** *key* Use the encryption *key* specified.

**–s** Selects software implementation for the encryption algorithm.

Two standard encryption modes are supported by the **des** program, Cipher Block Chaining ( CBC - the default) and Electronic Code Book ( ECB - specified with **-b**). CBC mode treats an entire file as a unit of encryption, that is, if insertions or deletions are made to the encrypted file then decryption will not succeed. CBC mode also ensures that regularities in clear data do not appear in the encrypted data. ECB mode treats each 8 bytes as units of encryptions, so if parts of the encrypted file are modified then other parts may still be decrypted. Identical values of clear text encrypt to identical values of cipher text.

The key used for the DES algorithm is obtained by prompting the user unless the **-k** *key* option is given. If the key is an argument to the **des** command, it is potentially visible to users executing **ps**(1) or a derivative. To minimize this possibility, **des** takes care to destroy the key argument immediately upon entry.

The **des** command attempts to use DES hardware for its job, but will use a software implementation of the DES algorithm if the hardware is unavailable. Normally, a warning message is printed if the DES hardware is unavailable since the software is only about 1/50th as fast. However, the **-f** option will suppress the warning. The **-s** option may be used to force use of software instead of hardware DES.

The **des** command reads from standard input unless *infile* is specified and writes to standard output unless *outfile* is given.

The following sections give information required to implement compatible facilities in other environments.

Since the CBC and ECB modes of DES require units of 8 bytes to be encrypted, files being encrypted by the **des** command have 1 to 8 bytes appended to them to cause them to be a multiple of 8 bytes. The last byte, when decrypted, gives the number of bytes (0 to 7) which are to be saved of the last 8 bytes. The other bytes of those appended to the input are randomized before encryption. If, when decrypting, the last byte is not in the range of 0 to 7 then either the encrypted file has been corrupted or an incorrect key was provided for decryption and an error message is printed.

The DES algorithm requires an 8 byte key whose low order bits are assumed to be odd-parity bits. The ASCII key supplied by the user is zero padded to 8 bytes and the high order bits are set to be odd-parity bits. The DES algorithm then ignores the low bit of each ASCII character, but that bit's information has been preserved in the high bit due to the parity.

The CBC mode of operation always uses an initial value of all zeros for the initialization vector, so the first 8 bytes of a file are encrypted the same whether in CBC or ECB mode.

FILES

**/dev/des?**

SEE ALSO

**ps**(1)

**BUGS**

It would be better to use a real 56-bit key rather than an ASCII -based 56-bit pattern. Knowing that the key was derived from ASCII radically reduces the time necessary for a brute-force crytographic attack.

**RESTRICTIONS**

Software encryption is disabled for programs shipped outside of the U.S. The program will still be able to encrypt files if one can obtain an encryption chip, legally or otherwise.

NAME

df – report free disk space on file systems

SYNOPSIS

**df** [ **–a** ] [ **–i** ] [ **–t** *type* ] [ *filesystem*... ] [ *filename*... ]

DESCRIPTION

**df** displays the amount of disk space occupied by currently mounted file systems, the amount of used and available space, and how much of the file system's total capacity has been used. Used without arguments, **df** reports on all mounted file systems, producing something like:

```
tutorial% df
Filesystem  kbytes  used  avail  capacity  Mounted on
/dev/ip0a    7445   4714  1986   70%       /
/dev/ip0g   42277  35291  2758   93%       /usr
```

Note that used+avail is less than the amount of space in the file system (kbytes); this is because the system reserves a fraction of the space in the file system to allow its file system allocation routines to work well. The amount reserved is typically about 10%; this may be adjusted using **tunefs**(8). When all the space on a file system except for this reserve is in use, only the super-user can allocate new files and data blocks to existing files. When a file system is overallocated in this way, **df** may report that the file system is more than 100% utilized.

If arguments to **df** are disk partitions (for example, **/dev/ip0a**s or path names, **df** produces a report on the file system containing the named file. Thus **df .** shows the amount of space on the file system containing the current directory.

OPTIONS

**–a** Reports on all filesystems including the uninteresting ones which have zero total blocks. (e.g. *automounter* )

**–i** Report the number of used and free inodes.

**–t** *type* Report on filesystems of a given *type* (for example, **nfs** or **4.2**).

FILES

**/etc/mtab** List of filesystems currently mounted.

SEE ALSO

**du**(1V), **mtab**(5), **quot**(8), **tunefs**(8)

NAME

diff – display line-by-line differences between pairs of text files

SYNOPSIS

**diff** [ **–bitw** ] [ **–c** [ # ] | **–e** | **–f** | **–n** | **–h** ] *filename1 filename2*
**diff** [ **–bitw** ] [ **–D***string* ] *filename1 filename2*
**diff** [ **–bitw** ] [ **–c** [ # ] | **–e** | **–f** | **–n** | **–h** ] [ **–l** ] [ **–r** ] [ **–s** ] [ **–S***name* ] *directory1 directory2*

DESCRIPTION

**diff** is a differential file comparator. When run on regular files, and when comparing text files that differ during directory comparison (see the notes below on comparing directories), **diff** tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, **diff** finds a smallest sufficient set of differences. If neither *filename1* nor *filename2* is a directory, either may be given as '–', in which case the standard input is used. If *filename1* is a directory, a file in that directory whose filename is the same as the filename of *filename2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

*n1* **a** *n3*,*n4*
*n1*,*n2* **d** *n3*
*n1*,*n2* **c** *n3*,*n4*

These lines resemble **ed**(1) commands to convert *filename1* into *filename2*. The numbers after the letters pertain to *filename2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *filename2* into *filename1*. As in **ed**(1), identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by '**<**', then all the lines that are affected in the second file flagged by '**>**'.

If both arguments are directories, **diff** sorts the contents of the directories by name, and then runs the regular file **diff** program as described above on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed.

OPTIONS

**–b** Ignore trailing blanks (SPACE and TAB characters) and treat all other strings of blanks as equivalent.

**–i** Ignore the case of letters; for example, '**A**' will compare equal to '**a**'.

**–t** Expand TAB characters in output lines. Normal or **–c** output adds character(s) to the front of each line which may screw up the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

**–w** Ignore all blanks (SPACE and TAB characters); for example, '**if ( a == b )**' will compare equal to '**if(a==b)**'.

The following four options are mutually exclusive:

**–c**[#] Produce a listing of differences with lines of context. The default is to present 3 lines of context and may be changed, (to 10, for example), by **–c10**. With **–c** the output format is modified slightly: output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen * s. The lines removed from *filename1* are marked with '– '; those added to *filename2* are marked '+ '. Lines which are changed from one file to the other are marked in both files with '! '.

Changes which lie within <context> lines of each other are grouped together on output. (This is a change from the previous '**diff -c**' but the resulting output is usually much easier to interpret.)

**–e** Produce a script of **a**, **c**, and **d** commands for the editor **ed**, which will recreate *filename2* from *filename1*.

In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version **ed** scripts ($2, $3, ...) made by **diff** need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Extra commands are added to the output when comparing directories with **–e**, so that the result is a **sh** script for converting text files which are common to the two directories from their state in *directory1* to their state in *directory2*.

**–f** Produce a script similar to that of **–e**, not useful with **ed**, which is in the opposite order.

**–n** Produce a script similar to that of **–e**, but in the opposite order and with a count of changed lines on each insert or delete command.

**–h** Do a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

Options for the second form of **diff** are as follows:

**–D***string*
Create a merged version of *filename1* and *filename2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *filename1*, while defining *string* will yield *filename2*.

Options when comparing directories are:

**–l** Long output format; each text file **diff** is piped through **pr**(1V) to paginate it, other differences are remembered and summarized after all text file differences are reported.

**–r** Apply **diff** recursively to common subdirectories encountered.

**–s** Report files which are the same, which are otherwise not mentioned.

**–S***name*
Start a directory **diff** in the middle, beginning with file *name*.

FILES

| | |
|---|---|
| **/tmp/d?????** | |
| **/usr/lib/diffh** | for **–h** |
| **/usr/bin/diff** | for directory diffs |
| **/usr/bin/pr** | |

SEE ALSO

**cc**(1V), **cmp**(1), **comm**(1), **cpp**(1), **diff3**(1V), **ed**(1), **pr**(1V)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

**Missing newline at end of file***X*
Indicates that the last line of file *X* did not have a NEWLINE. If the lines are different, they will be flagged and output, although the output will seem to indicate they are the same.

BUGS

Editing scripts produced under the **–e** or **–f** option are naive about creating lines consisting of a single '.'.

When comparing directories with the **–b**, **–w**, or **–i** options specified, **diff** first compares the files (as in **cmp**(1), and then runs the regular **diff** algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

The **–D** option ignores existing preprocessor controls in the source files, and can generate **#ifdefs**'s with overlapping scope. The output should be checked by hand, or run through '**cc –E**' (see **cc**(1V)) and then **diff**ed with the original source files. Discrepancies revealed should be corrected before compilation.

**NAME**

diff3 – display line-by-line differences between 3 files

**SYNOPSIS**

**diff3** [ **–exEX3** ] *filename1 filename2 filename3*

**SYSTEM V SYNOPSIS**

**/usr/5bin/diff3** [ **–ex3** ] *filename1 filename2 filename3*

**DESCRIPTION**

**diff3** compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

| | |
|---|---|
| **====** | All three files differ |
| **====1** | *filename1* is different |
| **====2** | *filename2* is different |
| **====3** | *filename3* is different |

The types of differences between a given range within the given files are indicated in one of these ways:

| | |
|---|---|
| *f*: *n1* **a** | Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3. |
| *f*: *n1*, *n2* **c** | Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*. |

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

**OPTIONS**

The options to **diff3** instruct it to produce a script for the editor **ed**, rather than a list of differences. This script will incorporate some or all of the differences between *filename2* and *filename3* into *filename1*. This script will not include a **w** or **q** command at the end, so that it will not write out the changed file.

**–e** Produce a script that will incorporate all changes between *filename2* and *filename3*, that is, the changes that normally would be flagged '====' and '====3'.

**–x** Produce a script that will incorporate only changes flagged '====='.

**–3** Produce a script that will incorporate only changes flagged '====3'.

**–E** Produce a script that will incorporate all changes between *filename2* and *filename3*, but treat overlapping changes (that is, changes that would be flagged with ==== in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by <<<<<< and >>>>>> lines.

**–X** Produce a script that will incorporate only changes flagged ====, but treat these changes in the manner of the **–E** option.

For example, suppose lines 7-8 are changed in both *filename1* and *filename2*. Applying the edit script generated by the command

**diff3 –E** *filename1 filename2 filename3*

to *filename1* results in the following file.

**lines 1-6**
**of** *filename1*
**<<<<<<<** *filename1*
**lines 7-8**
**of filename1**
**=======**
**lines 7-8**
**of** *filename3*
**>>>>>>>** *filename3*
**rest of** *filename1*

SYSTEM V OPTIONS

The System V version of **diff3** does not support the **–E** and **–X** options. The script produced by the **–e**, **–x**, and **–3** options *does* include a **w** and **q** command at the end, so that it *will* write out the changed file.

EXAMPLES

The following command will incorporate all the changes between *filename2* and *filename3* into *filename1*, and print the resulting file to the standard output. If the System V version of **diff3**, is used, *filename1* will be replaced with the resulting file.

**(diff3 –e** *filename1 filename2 filename3* **; echo '1,$p' )| ed –** *filename1*

FILES

**/tmp/d3?????**
**/usr/lib/diff3**
**/usr/5lib/diff3prog**

SEE ALSO

**diff**(1), **ed**(1)

BUGS

Text lines that consist of a single '.' will defeat a **–e** option.

NAME

diffmk – mark differences between versions of a troff input file

SYNOPSIS

**diffmk** *oldfile newfile markedfile*

AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**diffmk** compares two versions of a file and creates a third version that includes "change mark" (**.mc**) commands for **nroff**(1) and **troff**(1). *oldfile* and *newfile* are the old and new versions of the file. **diffmk** generates *markedfile*, which, contains the text from *newfile* with **troff**(1) "change mark" requests (**.mc**) inserted where *newfile* differs from *oldfile*. When *markedfile* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

**diffmk** can also be used in conjunction with the proper **troff** requests to produce program listings with marked changes. In the following command line:

**diffmk old.c new.c marked.c ; nroff**

the file **reqs** contains the following **troff** requests:

**.pl 1**
**.ll 77**
**.nf**
**.eo**
**.nh**

which eliminate page breaks, adjust the line length, set no-fill mode, ignore escape characters, and turn off hyphenation, respectively.

If the characters | and * are inappropriate, you might run *markedfile* through **sed**(1V) to globally change them.

SEE ALSO

**diff**(1), **nroff**(1), **sed**(1V), **troff**(1)

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, that is, replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

NAME

dircmp – compare directories

SYNOPSIS

**/usr/5bin/dircmp** [ **–d** ] [ **–s** ] [ **–w***n* ] *dir1 dir2*

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command. **dircmp** examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same contents.

OPTIONS

**–d** Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in **diff**(1).

**–s** Suppress messages about identical files.

**–w***n* Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

**cmp**(1), **diff**(1)

*Installing the SunOS*

NAME

dis – object code disassembler for COFF

SYNOPSIS

**dis** [ **–o** ] [ **–V** ] [ **–L** ] [ **–d** *sec* ] [ **–da** *sec* ] [ **–F** *function* ] [ **–t** *sec* ] [ **–l** *string* ] *coff-file* ...

AVAILABILITY

Sun386i systems only.

DESCRIPTION

The **dis** command produces an assembly-language listing of *coff-file*, which may be any object file in COFF format, or an archive of COFF object files.

The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

OPTIONS

**–o** Print numbers in octal. The default is hexadecimal.

**–V** Print, on standard error, the version number of the disassembler being executed.

**–L** Lookup source labels in the symbol table for subsequent printing. This option works only if the file was compiled with additional debugging information (e.g., the **–g** option of **cc**(1V)).

**–d** *sec* Disassemble the named section as data, printing the offset of the data from the beginning of the section.

**–da** *sec* Disassemble the named section as data, printing the actual address of the data.

**–F** *function*
Disassemble only the named function in each object file specified on the command line. The **–F** option may be specified multiple times on the command line.

**–t** *sec* Disassemble the named section as text.

**–l** *string*
Disassemble the library file specified by *string*. For example, **dis –l x –l z** disassembles **libx.a** and **libz.a**. All libraries are assumed to be in **/usr/lib**.

If the **–d**, **–da** or **–t** options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as **[5]**, represents that the breakpointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information (e.g., the **–g** option of **cc**(1V)). An expression such as **<40>** in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A function name will appear in the first column, followed by **( )**.

FILES

**/usr/lib**

SEE ALSO

**cc**(1V) **coff**(5)

NAME

domainname – set or display name of the current YP domain

SYNOPSIS

**domainname** [ *name-of-domain* ]

DESCRIPTION

Without an argument, **domainname** displays the name of the current domain, which typically encompasses a group of hosts under the same administration. As such, the name of a YP domain is normally also a valid Internet domain name, and can be used in conjunction with the **sendmail**(8) and the name server **named**(8C).

Only the super-user can set the name of the domain, by giving **domainname** an argument; this is usually done in the startup script **/etc/rc.local**.

SEE ALSO

**ypinit**(8), **named**(8C), **sendmail**(8)

## NAME

dos – SunView window for IBM PC/AT applications

## SYNOPSIS

**dos** [ **–b** ] [ **–s** ] [ **–p** *config* ] [ **–w** ] [ **–c** *command* ]

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

A window created by **dos** looks and acts like the screen of an IBM PC/AT or compatible computer running MS-DOS 3.3, except that it has expanded features. It allows sharing of files with SunOS, copying and pasting data between windows, and piping and redirection. You may run any reasonable number of DOS windows simultaneously.

Shrinking or expanding the window will not change the contents to accomodate the new size.

## USAGE

### Menu

The menu available in the window by pressing the right mouse button allows various controls over the work in the window. **Edit** allows you to copy and paste between windows. The **Show Screen** menu item selects the display type emulation (either Hercules, CGA, or Monochrome). The **Mouse** menu item allows you to control whether the mouse operates like a Microsoft or compatible mouse or in normal SunView fashion. The **Send to printer** menu item allows you to send queued jobs to the print spooler. **Device** allows you to select which disks and other devices will be used and which are to be considered write only. The **Reboot DOS Window** item is equivalent to restarting the window. This can also be accomplished by pressing the CONTROL, ALT, and DELETE keys simultaneously.

### Printer Assignments

DOS uses three printer designations: **LPT1**, **LPT2**, and **LPT3**. The default settings are: files sent to **LPT1** go to the default system printer. Files sent to **LPT2** are appended to the file **/pc/lpt2** in your home directory. Epson-compatible print jobs can be sent to **LPT3** to yield Epson FX-80 quality output on a Postscript printer.

### Drives

| | |
|---|---|
| Drive A | The Sun386i 3-1/2" diskette drive, used for reading PC format diskettes onto the hard disk and writing data to be stored on floppy. Drive A is not accessible across a network. |
| Drive C | A virtual disk stored in a file in */home/yourname/pc*. Files written to drive C cannot be accessed from SunOS. Drive C is generally intended for storage of applications and copy protected software but not data. |
| Drives D through S | Equivalents of SunOS directories. They can be accessed from either DOS or SunOS, and can contain any number of files and other directories. The SunOS directories referenced by DOS drives other than D, H, and R (described below) are user-defined (using the **DOS EXTEND** command). |
| Drive D | The current SunOS directory when the DOS window was opened. May subsequently be changed to any other directory. |
| Drive H | The home directory of the user who opened the window. May subsequently be changed to any directory in the user's home directory tree. |
| Drive R | Initially equivalent to the root directory of SunOS |

### File Sharing between SunOS and DOS

File names under DOS consist of 8 characters, a period, and a 3 character extension. When a SunOS filename does not comply with these rules, its name is modified by placing a tilde ( ) in an appropriate location so that the file name conforms to DOS specifications while remaining unique. It is recommended that filenames conform to DOS requirements for files to be used in both SunOS and DOS.

Because SunOS and DOS use different conventions for carriage returns, **dos2unix** and **unix2dos** are provided to convert text files between the two formats.

**Command Sharing between SunOS and DOS**

The **/etc/dos/unix** directory contains a list of SunOS commands accessible from DOS. Other SunOS commands not in this list can be executed from DOS with the command '**unix** *command*'. SunOS commands always use SunOS filename conventions and DOS commands always use DOS filename conventions, regardless of whether either type of command is executed from SunOS or DOS. Only DOS commands can use drives A and C.

OPTIONS

**–b** Boots (loads) DOS and opens a window using the **AUTOEXEC.BAT** and **CONFIG.SYS** files instead of **/pc/.quickpc**. A DOS sign-on message is displayed in the window.

**–s** Boot DOS and save a new **.quickpc** file under the name specified on the **SAVE** line in **/pc/.setup.pc**. Use this option after making changes to drive C **AUTOEXEC.BAT** or **CONFIG.SYS**. Exits DOS after saving the **.quickpc** file.

**–p** *config*
Loads an alternate file instead of **setup.pc**.

**–c** *command*
Executes the given DOS command in the newly created window.

**–w** Runs DOS text-only commands and applications in the current SunView Commands window.

ENVIRONMENT

**DOS_PRINTER** The value of this environment variable indicates the timeout (in seconds) for printing. A value of 20 (the default) indicates that jobs will be sent to the UNIX print spooler after 20 seconds of no printing activity from DOS to that printer. A value of 0 indicates that the spooler must be flushed manually from the menu in the window.

DOSLOOKUP
If on, this environment variable indicates that a command should be tried as a DOS command if not recognized by SunOS. If DOS supports the command, a DOS window is created and the command executed in that window. If the command does not exist, the normal SunOS error message results.

FILES

| | |
|---|---|
| **/etc/dos/unix** | Files in this directory indicate which SunOS commands are accessible from DOS. |
| **/etc/dos/defaults/.quickpc** | Default .quickpc file copied into user's home PC directory (/pc) the first time a DOS window is started. Not used by DOS in this location. |
| **/etc/dos/defaults/setup.pc** | Default setup.pc file copied into user's home DOS directory (/pc) the first time a DOS window is started. Not used by DOS in this location. |
| **/etc/dos/defaults/boards.pc** | Stores information about IBM PC/XT/AT-compatible boards installed in your system. |
| **/etc/dos/defaults/C:** | Default drive C file copied into a user's home PC directory the first time a DOS window is started. |
| **/pc/autoexec.bat** | Contains drive assignments, search paths, and other startup commands. Searched after **C:AUTOEXEC.BAT** and **D:AUTOEXEC.BAT**. |
| **C:AUTOEXEC.BAT** | Contains commands to access system printers and special drives. You should not need to change the **AUTOEXEC.BAT** on drive C. Put your changes in the **AUTOEXEC.BAT** on drive H (in your home directory). **C:AUTOEXEC.BAT** is not accessible from SunOS. |
| **D:AUTOEXEC.BAT** | If an **AUTOEXEC.BAT** file exists in the current directory, DOS tries execute faster running **C:AUTOEXEC.BAT**. |
| **C:CONFIG.SYS** | Specifies device drivers and other system parameters. C:CONFIG.SYS is not accessible from SunOS. |

| | |
|---|---|
| **/pc/setup.pc** | Defines printers, standard PC devices, and drive C. One or more of these files may exist, under various names which you assign. |
| **/pc/.quickpc** | An image of DOS as last saved with **dos -s**, including all DOS environment variables and drivers that were in effect at that time. DOS normally reads this file at startup. |
| **/pc/C:** | A user's personal copy of drive C. |

DIAGNOSTICS

**Cannot save** *filename* **quick-start file.**
The **dos** command was unable to save the specified quick-start file. Check the SAVE setting in your PC setup file (normally /pc/setup.pc) Also check file access permissions on the specified quick-start file.

**Cannot load** *filename* **quick-start file.**
**dos** was unable to read the specified quick-start file. Check the SAVE setting in your setup.pc file. Also check file access permissions on the specified quick-start file.

**Possible software incompatibility. Unsupported 286 instruction** *instruction* **at** *address*.
**Possible software incompatibility. Unsupported 386 instruction**
**Possible software incompatibility. Segment wrap.**
**Possible software incompatibility. Two-byte opcode not**
The application you are running was written specifically for 80286 or 80386 machines. Software run from a DOS window must be compatible with 8086 systems.

**Copying default configuration files into your**
This is the first time you have run the dos command. A **/pc** directory is being set up, and DOS-related files are being copied into it.

**Another DOS window already has access to** *device*
Your PC configuration file (normally **/pc/setup.pc**) is requesting access to a physical device that another DOS window is using.

**Port number** *number* **out of range for** *board* **board.**
The port number specified in the **/etc/dos/defaults/boards.pc** is invalid.

**Second port number** *number* **out of range for** *board* **board.**
The port number specified in the **/etc/dos/defaults/boards.pc** is invalid.

**IRQ value** *number* **out of range for** *board* **board.**
The interrupt level specified in the **/etc/dos/defaults/boards.pc** is invalid.

**Interrupt level** *number* **is used by DOS to support the** *device*
The interrupt level specified in the **/etc/dos/defaults/boards.pc** conflicts with an interrupt value currently being used by either a physical or emulated DOS device.

**I/O address range** *address* **-** *address* **requested for** *board already in use by device*.
The address range specified in the **/etc/dos/defaults/boards.pc** conflicts with range currently being used by either a physical or emulated DOS device.

**Cannot share** *device* **with a hardware interrupt.**
A shared device specified in the **/etc/dos/defaults/boards.pc** was also assigned an interrupt level in this file. Shared devices cannot be assigned interrupt levels.

**Couldn't find** *board* **in boards.pc.**
A file specified in the PC setup file (normally **/pc/setup.pc**) is not listed in the **/etc/dos/defaults/boards.pc** file. Check the **setup.pc** file, or add an entry for the board in **boards.pc**.

SEE ALSO

**dos2unix**(1), **unix2dos**(1)
*Sun386i User's Guide*
*Sun386i Advanced Skills*
*DOS Reference Manual*

NAME

dos2unix – convert text file from DOS format to SunOS format

SYNOPSIS

**dos2unix** [ **–iso** ] [ **–7** ] *originalfile convertedfile*

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**dos2unix** removes extra carriage returns and converts end of file characters in DOS format text files to conform to SunOS requirements.

This command can be invoked from either DOS or SunOS. However, the filenames must conform to the conventions of the environment in which the command is invoked.

If the original file and the converted file are the same, **dos2unix** will rewrite the original file after converting it.

OPTIONS

**–iso** Convert characters in the DOS extended character set to the corresponding ISO standard characters.

**–7** Convert 8 bit DOS graphics characters to 7 bit space characters so that SunOS can read the file.

DIAGNOSTICS

**File *filename* not found, or no read permission**

The input file you specified does not exist, or you do not have read permission (check with the SunOS **ls –l** command).

**Bad output filename *filename*, or no write permission**

The output file you specified is either invalid, or you do not have write permission for that file or the directory that contains it. Check also that the drive or diskette is not write-protected.

**Error while writing to temporary file**

An error occurred while converting your file, possibly because there is not enough space on the current drive. Check the amount of space on the current drive using the DIR command. Also be certain that the default diskette or drive is write-enabled (not write-protected). Note that when this error occurs, the original file remains intact.

**Could not rename temporary file to**
**Translated temporary file name = *filename*.**

The program could not perform the final step in converting your file. Your converted file is stored under the name indicated on the second line of this message.

SEE ALSO

*Sun386i Advanced Skills*
*DOS Reference Manual*

NAME

du – display the number of disk blocks used per directory or file

SYNOPSIS

**du** [ **–s** ] [ **–a** ] [ *filename* ... ]

SYSTEM V SYNOPSIS

**du** [ **–s** ] [ **–a** ] [ **–r** ] [ *filename* ... ]

DESCRIPTION

**du** gives the number of kilobytes contained in all files and, recursively, directories within each specified directory or file *filename*. If *filename* is missing, '.' (the current directory) is used.

A file which has multiple links to it is only counted once.

SYSTEM V DESCRIPTION

The System V version of **du** gives the number of 512-byte blocks rather than the number of kilobytes.

OPTIONS

**–s** Only display the grand total for each of the specified *filename*s.

**–a** Generate an entry for each file.

Entries are generated only for each directory in the absence of options.

SYSTEM V OPTIONS

**–r** The System V version of **du** is normally silent about directories that cannot be read, files that cannot be opened, etc. The **–r** option will cause **du** to generate messages in such instances.

EXAMPLE

Here is an example of using **du** in a directory. We used the **pwd**(1) command to identify the directory, then used **du** to show the usage of all the subdirectories in that directory. The grand total for the directory is the last entry in the display:

```
% pwd
/usr/ralph/misc
% du
5       ./jokes
33      ./squash
44      ./tech.papers/lpr.document
217     ./tech.papers/new.manager
401     ./tech.papers
144     ./memos
80      ./letters
388     ./window
93      ./messages
15      ./useful.news
1211    .
%
```

SEE ALSO

**df**(1), **pwd**(1), **quot**(8)

BUGS

Filename arguments that are not directory names are ignored, unless you use **–a** .
If there are too many distinct linked files, **du** will count the excess files more than once.

NAME

echo – echo arguments to the standard output

SYNOPSIS

**echo** [ **–n** ] [ *argument* ... ]

SYSTEM V SYNOPSIS

**echo** *argument* ...

DESCRIPTION

**echo** writes its arguments on the standard output. Arguments must be separated by SPACE characters or TAB characters, and terminated by a NEWLINE.

**echo** is useful for producing diagnostics in shell programs and for writing constant data on pipes. If you are using the Bourne shell ( **sh**(1)), you can send diagnostics to the standard error file by typing:

**echo ... 1>&**

SYSTEM V DESCRIPTION

Note: If **/usr/5bin** is ahead of **/usr/bin** in the Bourne shell's search path, its built-in **echo** command mimics the System V version of **echo** as described here.

**echo** also understands C –like escape conventions; beware of conflicts with the shell's use of '\':

| | |
|---|---|
| **\b** | BACKSPACE |
| **\c** | Print line without NEWLINE |
| **\f** | FORMFEED |
| **\n** | NEWLINE |
| **\r** | RETURN |
| **\t** | TAB |
| **\v** | vertical TAB |
| \\ | backslash |
| *\n* | the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero. |

OPTIONS

| | |
|---|---|
| **–n** | Do not add the NEWLINE to the output. |

FILES

**/usr/5bin**
**/usr/bin**

SEE ALSO

**sh**(1)

## NAME

ed – basic line editor

## SYNOPSIS

**ed** [ – ] [ **–sx** ] [ **–p** *string* ] [ *filename* ]

## DESCRIPTION

**ed** is the most basic line editor of the UNIX system. Although superseded by **ex**(1) and **vi**(1) for most purposes, **ed** is still used by various system utilities.

**ed** operates on a copy of *filename*, called a buffer, and overwrites a file only when you issue the **w** (write) command. **ed** provides line oriented editing commands to display or change lines, to insert and delete lines from the buffer, to move or copy lines within the buffer, or to substitute character strings within lines.

## OPTIONS

–

**–s** Suppress printing of character counts (by **e**, **r**, and **w** commands), diagnostics (by **e** and **q** commands), and the **!** prompt (after a **!** command). Also, suppress printing the **?** diagnostic before overwriting unsaved changes in the buffer.

**–x** Edit an encrypted file (see **crypt**(1) for details).

**–p** *string* Use *string* as the editing prompt in command mode.

## USAGE

### Command Structure

**ed** commands have a simple and regular structure. They consist of an optional *address*, or two optional *address*es separated by a comma or semicolon, then a single-character *command*, which may be followed by a *parameter* for that *command*:

[*address*[ ,*address* ]] *command* [*parameter* ]

If only one *address* is specified, operations are performed on that line. If two *address*es are specified, **ed** performs the operation on the inclusive range of lines. Commands that requires an *address* use certain addresses by default, typically the address of the current line.

For example, **1,10p** means "print (display) lines 1 through 10" (two addresses), **5a** means "append text after line 5" (one address), and **d** means "delete the current line" (no address with the current line used as default). The meaning of *parameter* varies for each operation — for the move (**m**) and transfer (**t**) operations, for instance, it is the line that the addressed lines are to be moved to or transferred after. For reading (**r**) and writing (**w**) a file, *parameter* specifies the name of the file that is to be read or written.

**ed** is extremely terse in its interaction with the user. Its normal response to most problems is simply a question mark (**?**). This may happen when **ed** cannot find a specified line in the buffer, or if a search for a regular expression fails in a substitute (**s**) command. The **h** command prints a somewhat more complete diagnostic for the most recent error encountered; the **H** command requests that the diagnostic be printed for all errors.

### Addresses

Lines can be addressed in several ways:

*nnn* By line number. Lines in the buffer are numbered relative to the start of the buffer. When displayed, line numbers are not physically present with the text of the file or buffer.

**$** The last line of the buffer.

**.** The current line. **ed** keeps track of the line on which you last performed an operation. This line is called the *current line*. You can address this line by typing a "dot" character.

*±n* By relative line number. Address the line number that is *n* lines higher, or *n* lines lower than the current line.

*'c* Address the line marked with the mark character *c*, which must be a lower-case letter. Lines are marked with the **k** command, described below.

*/RE/* An *RE* is a Regular Expression, described under **Regular Expressions** below. When enclosed by slashes, *RE* addresses the first line found by searching for a matching string. The search proceeds forward from the line following the current line, and wraps through the beginning of the buffer to include all preceding lines, as well as the current line.

*?RE?* An *RE* enclosed in question marks addresses the first line containing a match found by searching backward from the line preceding the current line. The search wraps through the end of the buffer to include all lines following the current line (in reverse order), as well as the current line.

*address±n*

An address followed by a plus sign (+) or a minus sign (–), followed by a decimal number, specifies that address plus or minus the indicated number of lines. (The plus sign may be omitted.) If the address is omitted, the current line is used as the base. For example, '**31–3**' addresses line 28 in the buffer.

*address±*

If an address ends with '+' or '–', then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and the previous rule, the address '–' refers to the line preceding the current line. (To maintain compatibility with earlier versions of **ed**, the character '^' is equivalent to '–'.) Trailing '+' and '–' characters have a cumulative effect, so '––' refers to the current line, less 2.

, By itself, a comma stands for the address pair '**1,$**'.

; A semicolon by itself stands for the pair '**.,$**'.

By default for a given command. If you do not specify an address for a command to operate on, a command that requires an address supplies one by default. This is typically the current line.

A pair of addresses separated by a comma signifies an inclusive range of lines, and the current line is not changed unless the command changes it. When addresses are separated by a semicolon, however, the current line is set to the address preceding the semicolon before any subsequent addresses are interpreted. This feature can be used to determine the starting line for forward and backward searches using '/', and '?'.

The second address of any two-address sequence must correspond to a line that occurs later in the buffer than that of the first address.

**Regular Expressions**

**ed** supports a limited form of regular-expression notation, which can be used in a line address to specify lines by content. A regular expression (RE) specifies a set of character strings to match against — such as "any string containing digits 5 through 9" or "only lines containing uppercase letters." A member of this set of strings is said to be *matched* by the regular expression. Regular expressions or *patterns* are used to address lines in the buffer (see **Addresses** , above), and also for selecting strings to be replaced using the **s** (substitute) command.

Where multiple matches are present in a line, a regular expression matches the the *longest* of the *leftmost* matching strings.

Regular expressions can be built up from the following "single-character" RE's:

*c* Any ordinary character not listed below. An ordinary character matches itself.

\ Backslash. When followed by a special character, the RE matches the "quoted" character. A backslash followed by one of <, >, (, ), {, or }, represents an *operator* in a regular expression, as described below.

. Dot. Matches any single character except NEWLINE.

^ As the leftmost character, a caret (or circumflex) constrains the RE to match the leftmost portion of a line. A match of this type is called an "anchored match" because it is "anchored" to a specific place in the line. The ^ character loses its special meaning if it appears in any position other than the start of the RE.

$ As the rightmost character, a dollar sign constrains the RE to match the rightmost portion of a line. The $ character loses its special meaning if it appears in any position other than at the end of the RE.

^*RE*$ The construction ^*RE* $ constrains the RE to match the entire line.

\< The sequence \< in an RE constrains the one-character RE immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> The sequence \> in an RE constrains the one-character RE immediately following it only to match something at the end of a "word."

[ *c* . . . ] A nonempty string of characters, enclosed in square brackets matches any single character in the string. For example, [abcxyz] matches any single character from the set '**abcxyz**'. When the first character of the string is a caret (^), then the RE matches any character *except* NEWLINE and those in the remainder of the string. For example, '[^45678]' matches any character except '**45678**'. A caret in any other position is interpreted as an ordinary character.

[ ] *c* . . . ] The right square bracket does not terminate the enclosed string if it is the first character (after an initial '^', if any), in the bracketed string. In this position it is treated as an ordinary character.

[ *l*–*r* ] The minus sign, between two characters, indicates a range of consecutive ASCII characters to match. For example, the range '[0–9]' is equivalent to the string '[0123456789]'. Such a bracketed string of characters is known as a *character class*. The '–' is treated as an ordinary character if it occurs first (or first after an initial ^) or last in the string.

*d* Delimiter character. The character used to delimit an RE within a command is special for that command (for example, see how / is used in the **g** command, below).

The following rules and special characters allow for constructing RE's from single-character RE's:

A concatenation of RE's matches a concatenation of text strings, each of which is a match for a successive RE in the search pattern.

* A single-character RE, followed by an asterisk (*) matches *zero* or more occurrences of the single-character RE. Such a pattern is called a *closure*. For example, [a–z][a–z]* matches any string of one or more lower case letters.

\{m\}
\{m,\}
\{m,n\} A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is an RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be nonnegative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n*, inclusively. Whenever a choice exists, the RE matches as many occurrences as possible.

\(. . .\) An RE enclosed between the character sequences \( and \) matches whatever the unadorned RE matches, but saves the string matched by the enclosed RE in a numbered substring register. There can be up to nine such substrings in an RE, and parenthesis operators can be nested.

*n* Match the contents of the *n*th substring register from the current RE. This provides a mechanism for extracting matched substrings. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string. When nested parenthesized substrings are present, *n* is determined by counting occurrences of \( starting from the left.

// The null RE (//) is equivalent to the last RE encountered.

**Commands**

The commands **a** for *append*, **c** for *change*, and **i** for *insert*, allow you to add new text to the buffer. While accepting new text, **ed** is said to be in *input mode*. While in input mode, *no* commands are recognized; all character input is inserted into the buffer. To exit from input mode, enter a dot (**.**) on a line by itself; **ed** then reverts to command mode. Or, you can interrupt **ed** (typically with CTRL-C), in which case it displays

a **?** and returns to command mode.

Commands may accept zero, one, or two addresses. Commands that accept no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when too few addresses are given; if more addresses are given than such a command requires, only the last ones are used.

In the following list of **ed** commands, the default addresses are shown in parentheses; the parenthesized addresses are *not* part of the command.

It is generally illegal for more than one command to appear on a line. However, any command (except **e**, **f**, **r**, or **w**) may be followed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively.

(.)**a**
*text*
.

Append text. Add lines of *text* into the buffer after the addressed line. The resulting current line is the last line of input, or the addressed line if no text is entered. Address **0** is legal for this command, in which case the *text* is placed at the beginning of the buffer. The maximum number of characters per input line (from a terminal) is 256, including the final NEWLINE.

(.)**c**
*text*
.

Change lines. Delete the addressed lines, and then accept lines of *text* to replace them. **c** accepts one or two addresses; the default is the current line. The resulting current line is the last line of input, or the line preceding the deleted lines if no text is entered.

(.,.)**d** Delete the addressed lines from the buffer. **d** accepts one or two addresses; the default is the current line. The resulting current line is the line following the last one deleted; if the deleted lines were at the end of the buffer, the new last line is the resulting current line.

**e** *filename*

Edit a file. Delete the entire contents of the buffer, and then read in the named file. The resulting current line is the last line of the buffer. **e** reports the number of characters read into the buffer, and sets *filename* to be the current file (for use as a default filename in subsequent commands). If no *filename* is given, the current filename, if any, is used (see the **f** command, below). If *filename* is replaced by a shell (**sh**(1)) command prefaced with a '**!**', the shell command is executed and its output is read into the buffer after the current line. Such a shell command is *not* used as the current filename. **e** displays a **?** if the buffer has not been written out since the last change made — another **e** command in response to the **?** forces the command to take effect.

**E** *filename*

The **E** command is like **e**, except that the editor does not check for changes to the buffer since the last **w** command was performed.

**f** *filename*

Display or set the current filename. If *filename* is given as an argument, the file (**f**) command changes the current filename to *filename*; otherwise, it prints the current filename.

(1,$) **g**/*RE*/*command-list*

The global (**g**) command performs *command-list* on all lines in the range of addresses that match *RE*. **ed** executes *command-list* for each matching line in succession, setting the current line to each in turn. *command-list* can contain a single command, or it can be continued across input lines, with one **ed** command per line, by escaping all but the last NEWLINE with a \ character. Operations that place **ed** into input mode (**a**, **i**, and **c**), *are* permitted in *command-list*; the final '**.**' terminating text input may be omitted if it is the last line of the *command-list*. **g**, **G**, **v**, and **V** commands, however, are *not* permitted. An empty *command-list* is equivalent to the **p** command.

(1,$) **G**/*RE*/

The interactive **G** (Global) command, selects all lines that match the given *RE*. Then, each selected line is made current, and any *one* command (other than one of the **a, c, i, g, G, v,** and **V** commands) can be performed upon that line. A NEWLINE acts as a null command; an **&** reexecutes the most recent command. Commands entered during execution of the **G** command can address and affect lines other than the current line. The **G** command can be terminated by an interrupt (typically CTRL-D).

**h** Help. Display a short error message that explains the reason for the most recent **?** diagnostic.

**H** Automatic printing of help diagnostics. Toggle between printing the **?** diagnostic, or automatically printing diagnostic messages as well.

(.)**i**
*text*
.

Insert Text. Insert the given *text* into the buffer, above the addressed line. **i** accepts one *address*; the default is the current line. The resulting current line is the last line of input; if no text is input, it is the line just before the addressed line. This command differs from the *a* command only in the placement of the input text; Address 0 is not allowed for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the NEWLINE character).

(.,.+1)**j** Join Lines. Remove the NEWLINE character from between the two addressed lines. The defaults are the current line and the line following. If exactly one address is given, this command does nothing. The joined line is the resulting current line.

(.)**k***c* Mark the addressed line with the name *c*, a lower-case letter. The address-form, *'c*, addresses the line marked by *c*. **k** accepts one *address*; the default is the current line. The current line is left unchanged.

(.,.)**l** List nonprinting characters. Print the addressed lines in an unambiguous way: a few nonprinting characters, such as TAB and BACKSPACE are represented by visually mnemonic overstrikes. All other nonprinting characters are shown in octal, with long lines folded. **l** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. An **l** command may be appended to any command other than **e, f, r,** or **w.**

(.,.) **m***address*

Move addressed lines to just after *address*. Address 0 is legal, and moves the addressed line(s) to the beginning of the file. An error results if *address* falls within the range of lines to move. **m** accepts two addresses to specify a range of lines to move; the default is the current line. The resulting current line is the last of the moved lines.

(.,.)**n** Number the displayed lines. Print the addressed lines, preceding each with its line number and a TAB character. **n** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **n** command can be appended to any command other than **e, f, r,** or **w.**

(.,.)**p** Print the addressed lines. **p** accepts one or two addresses; the default is the current line. The resulting current line is the last line printed. The **p** command may be appended to any command other than **e, f, r,** or **w.** For example, **dp** deletes the current line and prints the new current line.

**P** Toggle prompting on or off. When prompting is in effect, the editor prompts with a * for commands. A subsequent **P** command turns prompting off.

**q** Quit. Exit from **ed.** Note, however, that the buffer is *not* automatically written out; you must write any changes to be saved with the **w** command; **ed** warns you once if you have not saved your changes (unless the '–' option is in effect). A second **q** forces **ed** to exit regardless, destroying the buffer's contents.

**Q** Force quit. This is the same as **q**, but you do not get any warning if you have not previously written out the buffer. **ed** simply exits.

(**$**) **r** *filename*

Read in the contents of *filename*, after the addressed line. If *filename* is not given, the current filename, if any, is used (see the **e** and **f** commands). The current filename is *not* altered; if there is no current filename, *filename* becomes the current filename. **r** accepts one *address*; the default is **$**. Address 0 is legal for **r**, in which case the file is read in at the beginning of the buffer. If the read is successful, the number of characters read is typed. The resulting current line is the last line read in from the file. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a **!**, the shell command is executed and its output is read in. Such a shell command is *not* remembered as the current filename.

(.,.) **s**/*RE*/*rs*/
(.,.) **s**/*RE*/*rs*/**g**
(.,.) **s**/*RE*/*rs*/*n*

Substitute. Search each addressed line for the first occurrence of a string matching the specified *RE*, and replace it with *rs*, the replacement string. If **g** (global suffix) is appended to the command, replace *all* (non-overlapped) matching strings in each addressed line with the replacement string *rs*. Note: the **g** *suffix* is distinct from the **g** *command*. If a number *n* is appended, replace only the *n*'th occurrence of the matched string on each addressed line. **s** accepts one or two addresses; the default is the current line. The resulting current line is the last line on which a substitution is made. An error results if *RE* matches no strings in the addressed line or range. Any character (other than SPACE or NEWLINE can be used instead of / to delimit *RE* and *rs*. As with RE's in addresses, you can refer to the entire string matched by *RE* with an '**&**'; you can refer to parenthesized substrings within *RE* using **1**...**n**. When **%** is the only character in *rs*, the *rs* from in the most recent substitute command is used as the current *rs*. The **%** loses its special meaning when it is in a replacement string of more than one character, or if it is preceded by a backslash.

A line may be split by substituting a NEWLINE character into it. The NEWLINE in the *replacement* must be escaped by preceding with an '\'. Such substitutions cannot be done as part of a **g** or **v** command list.

(.,.) **t***address*

Transfer. Transpose a copy of the addressed range of lines to just after the given *address*. **t** (transfer) is like **m** (move), except that it copies of the lines, rather than moving them. **t** accepts two addresses preceding the operation letter, the current address is the default. The resulting current line is the last line copied. Address 0 is allowed.

**u** Undo. Reverse the effect of the most recent command that modified the buffer. A second **u** undoes the undo operation.

(**1**,**$**) **v**/*RE*/*command-list*

This command is the same as the global command **g** except that the *command-list* is executed with '**.**' initially set to every line that does *not* match the RE.

(**1**,**$**) **V**/*RE*

Similar to the **G** command, except that the lines selected are those that do *not* match the *RE*.

(**1**,**$**) **w** *filename*

Write the addressed lines to *filename*. If the file does not exist, **ed** creates it. The current filename is *not* altered; if there is no current filename, then *filename* becomes current. If no *filename* is given, the current filename, if any, is used. **w** accepts one or two addresses; the default is all lines in the file. The current line is unchanged. If the command is successful, the number of characters written is displayed. If *filename* is replaced by a shell (**sh**(1)) command prefaced with a '**!**', the shell command is executed with standard input taken from the addressed lines. Such a shell command is *not* remembered as the current filename.

(**1**,**$**) **W** *filename*

Like **w**, but append the addressed lines onto the named file.

**x** Encrypt the file. **ed** prompts for an encryption key from the standard input. Subsequent **e**, **r**, and

**w** commands encrypt and decrypt the text with this key (see **crypt**(1)). An empty key turns off encryption. Encryption can also be specified on the command line with the **–x** option.

(**$**)**=** Display the line number of the addressed line; the current line remains unchanged.

**!***shell-command*

Run a shell command. *shell-command* is a (Bourne shell) command line. **ed** replaces the unescaped character **%** with the current filename; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. (**!!** repeats the last shell command.) If any such expansion is performed, the expanded line is echoed. The current line is unchanged.

*address*

NEWLINE

An address, alone on a line, prints the addressed line. A NEWLINE alone is equivalent to **'.+1p'** which is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, **ed** prints a **?** and returns to *its* command level.

**File Format Specification Support**

**ed** supports the **fspec**(5) formatting capability for displaying lines. When the first line of a file is a format specification of the form:

**<:***ts*[**,***ts*]... **s***max***:>**

where *ts* is the column number of a tab stop and *max* is the maximum line length for display purposes, and with the terminal in **'stty –tabs'** or **'stty tab3'** mode (see **stty**(1V) for details), the indicated tab stops are used in displayed lines. While inserting text, however, tab stops are set to every eighth column.

## ENVIRONMENT

**TMPDIR** If this environment variable is set and is not null, its value is used in place of **/usr/tmp** as the directory in which the temporary file is placed.

## FILES

| | |
|---|---|
| **/usr/tmp/e#** | temporary; # is the process number |
| **ed.hup** | file for saved work if the terminal is hung up |

## SEE ALSO

**crypt**(1), **ex**(1), **grep**(1V), **sed**(1V), **sh**(1), **stty**(1V), **vi**(1), **regexp**(3), **fspec**(5)

*Editing Text Files*

## LIMITATIONS

The following limitations apply:

512 characters per line.
256 characters per global command-list.
1024 characters per filename.
The limit on the number of lines depends on the amount of user memory:
each line takes 1 word.

When reading a file, **ed** discards ASCII NUL characters and all characters after the last NEWLINE. Files (such as executable images) that contain characters not in the ASCII set (bit 8 on) cannot be edited using **ed**.

If a file is not terminated by a NEWLINE character, **ed** adds one and prints a message saying that it has done so.

If the closing delimiter of an RE or of a replacement string (such as /) would be the last character before a NEWLINE, that delimiter can be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

| | |
|---|---|
| **s/s1/s2** | **s/s1/s2/p** |
| **g/s1** | **g/s1/p** |
| **?s1** | **?s1?** |

## DIAGNOSTICS

**?** For command errors.

***?file*:*error***

For an inaccessible file (use the **h** (help) and **H** (Help) commands for detailed explanations).

If changes have been made in the buffer since the last **w** command, **ed** issues a warning **?** when a command is given that would destroy the buffers contents. A second **e** or **q** command at this point will take effect. The '–' and **–s** command-line options inhibit this feature.

## CAVEATS AND BUGS

A **!** command cannot be subject to a **g** or a **v** command.

The sequence **\n** in an RE does not match a NEWLINE character.

Files encrypted directly with the **crypt**(1) command with the null key cannot be edited.

The encryption facilities of **ed** are not available on software shipped outside the U.S.

Characters are masked to 7 bits on input.

If the editor input is coming from a command file, the editor exits at the first failure of a command in that file.

## NAME

env – obtain or alter environment variables for command execution

## SYNOPSIS

**env** [ – ] [ *name=value* ... ] [ *command* ]

## DESCRIPTION

**env** obtains the current **environment,** modifies it according to its arguments, and executes the command with the modified environment that results.

If no *command* is specified, the resulting environment is displayed.

## OPTIONS

| | |
|---|---|
| – | Ignore the environment that would otherwise be inherited from the current shell. Restricts the environment for *command* to that specified by the arguments. |
| *name=value* | Set the environment variable *filename* to *value* and add it to the environment. |

## SEE ALSO

**sh**(1), **execve**(2), **profil**(2), **environ**(5V)

NAME

eqn, neqn, checkeq – typeset mathematics

SYNOPSIS

**eqn** [ **–d***xy* ] [ **–p***n* ] [ **–s***n* ] [ **–f***n* ] [ *filename* ] ...
**neqn** [ *filename* ] ...
**checkeq** [ *filename* ] ...

AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**eqn** (and **neqn**) are language processors to assist in describing equations. **eqn** is a preprocessor for **troff**(1) and is intended for devices that can print **troff**'s output. **neqn** is a preprocessor for **nroff**(1) and is intended for use with terminals. Usage is almost always:

**tutorial% eqn** *file* ... | **troff**
**tutorial% neqn** *file* ... | **nroff**

If no *file*s are specified, **eqn** and **neqn** read from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as **eqn** input.

**checkeq** reports missing or unbalanced delimiters and .EQ/.EN pairs.

OPTIONS

**–d***xy* Set equation **d**elimiters set to characters *x* and *y* with the command-line argument. The more common way to do this is with **delim***xy* between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by **delim off** appearing in the text. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

**–p***n* Reduce subscripts and superscripts by *n* point sizes from the previous size. In the absence of the **–p** option, subscripts and superscripts are reduced by 3 point sizes from the previous size.

**–s***n* Change point size to *n* globally in the document. The point size can also be changed globally in the body of the document by using the **gsize** directive.

**–f***n* Change font to *n* globally in the document. The font can also be changed globally in the body of the document by using the **gfont** directive.

EQN LANGUAGE

Tokens within **eqn** are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ( ) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus **x sub i** makes $x_i$, **a sub i sup 2** produces $a_i^2$, and **e sup {x sup 2 + y sup 2}** gives $e^{x^2+y^2}$.

Fractions are made with **over**: **a over b** yields $\frac{a}{b}$.

**sqrt** makes square roots: **1 over sqrt {ax sup 2 +bx+c}** results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n \to \infty} \sum_{0}^{n} x_i$ is made with **lim from {n–> inf } sum from 0 to n x sub i**.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: **left [ x sup 2 + y sup 2 over alpha right ] = 1** produces $\left[ x^2 + \frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional. Legal characters after **left**

and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile, lpile, cpile,** and **rpile: pile {a above b above c}** produces $\begin{matrix} a \\ b \\ c \end{matrix}$.
There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix: matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }** produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$.
In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot, dotdot, hat, tilde, bar, vec, dyad,** and **under: x dot = f(t) bar** is $\dot{x} = \overline{f(t)}$, **y dotdot bar = n under** is $\bar{\ddot{y}} = \underline{n}$, and **x vec = y dyad** is $\vec{x} = \vec{\vec{y}}$.

Sizes and font can be changed with **size** *n* or **size** ±*n*, **roman, italic, bold,** and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments –s*n* and –f*n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

**define** *thing % replacement %*

defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords like **sum** ($\sum$), **int** ($\int$), **inf** ($\infty$), and shorthands like >= ($\geq$), -> ($\rightarrow$), and != ($\neq$) are recognized. Greek letters are spelled out in the desired case, as in **alpha** or **GAMMA**. Mathematical words like sin, cos, log are made Roman automatically. **troff**(1) four-character escapes like \(bu (•) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with **troff** when all else fails.

SEE ALSO

**troff**(1), **tbl**(1), **ms**(7), **eqnchar**(7)

*Formatting Documents*

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in **bold "12.3"**.

NAME

error – categorize compiler error messages, insert at responsible source file lines

SYNOPSIS

**error** [ **–n** ] [ **–s** ] [ **–q** ] [ **–v** ] [ **–t** *suffixlist* ] [ **–I** *ignorefile* ] [ *filename* ]

DESCRIPTION

**error** analyzes error messages produced by a number of compilers and language processors. It replaces the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously.

**error** looks at error messages, either from the specified file *filename* or from the standard input, and:

- Determines which language processor produced each error message.
- Determines the file name and line number of the erroneous line.
- Inserts the error message into the source file immediately preceding the erroneous line.

Error messages that can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. **error** touches source files only after all input has been read.

**error** is intended to be run with its standard input connected with a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into **error.** For example, when using the **csh** syntax,

**tutorial% make –s lint |& error**

will analyze all the error messages produced by whatever programs **make**(1) runs when making lint.

**error** knows about the error messages produced by: **make**(1), **cc**(1V), **cpp**(1), **as**(1), **ld**(1), **lint**(1V), and other compilers. For all languages except Pascal, error messages are restricted to one line. Some error messages refer to more than one line in more than one file, in which case **error** duplicates the error message and inserts it in all the appropriate places.

OPTIONS

**–n** Do *not* touch any files; all error messages are sent to the standard output.

**–q** **error** asks whether the file should be touched. A '**y**' or '**n**' to the question is necessary to continue. Absence of the **–q** option implies that all referenced files (except those referring to discarded error messages) are to be touched.

**–v** After all files have been touched, overlay the visual editor **vi** with it set up to edit all files touched, and positioned in the first touched file at the first error. If **vi**(1) can't be found, try **ex**(1) or **ed**(1) from standard places.

**–t** *suffixlist*

Take the following argument as a suffix list. Files whose suffices do not appear in the suffix list are not touched. The suffix list is dot separated, and '*' wildcards work. Thus the suffix list:

**.c.y.f*.h**

allows **error** to touch files ending with '**.c**', '**.y**', '**.f***' and '**.h**'.

**–s** Print out statistics regarding the error categorization.

**error** catches interrupt and terminate signals, and terminates in an orderly fashion.

USAGE

**Action Statements**

**error** does one of six things with error messages.

**synchronize** Some language processors produce short errors describing which file they are processing. **Error** uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by **error.**

**discard** Error messages from **lint** that refer to one of the two **lint** libraries, **/usr/lib/lint/llib-lc** and **/usr/lib/lint/llib-port** are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by **error.**

**nullify** Error messages from **lint** can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named **.errorrc** in the user's home directory, or from the file named by the **–I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

**not file specific** Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They are not inserted into any source file.

**file specific** Error messages that refer to a specific file but to no specific line are written to the standard output when that file is touched.

**true errors** Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are inserted into source files. Other error messages are consumed entirely by **error** or are written to the standard output. **error** inserts the error messages into the source file on the line preceding the line number in the error message. Each error message is turned into a one line comment for the language, and is internally flagged with the string **###** at the beginning of the error, and **%%%** at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

FILES

| | |
|---|---|
| **/.errorrc** | function names to ignore for **lint** error messages |
| **/dev/tty** | user's teletype |

SEE ALSO

**as**(1), **cc**(1V), **cpp**(1), **csh**(1), **ed**(1), **ex**(1), **ld**(1), **lint**(1V), **make**(1), **vi**(1)

BUGS

Opens the tty-device directly for user input.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's error message format may cause **error** to not understand the error message.

**error,** since it is purely mechanical, will not filter out subsequent errors caused by "floodgating" initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected, error puts them before. The alignment of the | marking the point of error is also disturbed by **error.**

**error** was designed for work on CRT 's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

## NAME

ex, edit, e – line editor

## SYNOPSIS

**ex** [ – ] [ –**lrRvx** ] [ –**t** *tag* ] [ –**w***nnn* ] [ +*command* ] *filename*...

**edit** [ *options* ]

## DESCRIPTION

**ex**, a line editor, is the root of a family of editors that includes **edit**, **ex**(1), and **vi**(1) (the display editor). In most cases **vi** is preferred for interactive use.

## OPTIONS

| | |
|---|---|
| – | Suppress all interactive feedback to the user (useful for processing **ex** scripts in shell files). |
| –**l** | Set up for editing LISP programs. |
| –**r** | Recover the indicated *filename*s after a system crash. |
| –**R** | Read only. Do not overwrite the original file. |
| –**v** | Start up in display editing state using **vi**. You can achieve the same effect by simply typing the **vi** command itself. |
| –**x** | Prompt for a key to be used in encrypting the file being edited. |
| –**t** *tag* | Edit the file containing the tag *tag*. A tags database must first be created using the **ctags**(1) command. |
| –**w***nnn* | Set the default window (number of lines on your terminal) to *nnn* — this is useful if you are dialing into the system over a slow phone line. |
| +*command* | Start the editing session by executing *command*. |

## ENVIRONMENT

The editor recognizes the environment variable **EXINIT** as a command (or list of commands separated by | characters) to run when it starts up. If this variable is undefined, the editor checks for startup commands in the file **/.exrc** file, which you must own. However, if there is a **.exrc** owned by you in the current directory, the editor takes its startup commands from this file — overriding both the file in your home directory and the environment variable.

## FILES

| | |
|---|---|
| **/usr/lib/ex?.?strings** | error messages |
| **/usr/lib/ex?.?recover** | recover command |
| **/usr/lib/ex?.?preserve** | preserve command |
| **/etc/termcap** | describes capabilities of terminals |
| **.exrc** | editor startup file for current directory |
| **/.exrc** | user's editor startup file |
| **/tmp/Ex***nnnnn* | editor temporary file |
| **/tmp/Rx***nnnnn* | file named buffer temporary |
| **/var/preserve** | preservation directory |

## SEE ALSO

**awk**(1), **ctags**(1), **ed**(1), **vi**(1), **grep**(1V), **sed**(1V), **termcap**(5), **environ**(5V)

*Editing Text Files*

## BUGS

All marks are lost on lines changed and then restored with the **undo** commmand, if the marked lines were changed. **undo** never clears the buffer modified condition.

The **z** command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '–' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

With the **modeline** option in effect, the editor checks the first five lines of the text file for commands of the form

**ex:** *command*:

or

**vi:** *command*:

if any are found, the editor executes them. This can result in unexpected behavior, and is not recommended in any case. In earlier releases, **modeline** was in effect by default. Now it is not, but setting it in the **.exrc** file or the **EXINIT** environment variable can still produce untoward effects.

## RESTRICTIONS

The encryption facilities of **ex** are not available on software shipped outside the U.S.

## NAME

expand, unexpand – expand TAB characters to SPACE characters, and vice versa

## SYNOPSIS

**expand** [ *–tabstop* ] [ *–tab1, tab2,..., tabn* ] [ *filename ...* ]

**unexpand** [ **–a** ] [ *filename ...* ]

## DESCRIPTION

**expand** copies *filename*s (or the standard input) to the standard output, with TAB characters expanded to SPACE characters. BACKSPACE characters are preserved into the output and decrement the column count for TAB calculations. **expand** is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain TAB characters.

**unexpand** copies *filename*s (or the standard input) to the standard output, putting TAB characters back into the data. By default, only leading SPACE and TAB characters are converted to strings of tabs, but this can be overridden by the **–a** option (see the OPTIONS section below).

## OPTIONS

**expand**

*–tabstop*

Specify as a single argument, sets TAB characters *tabstop* SPACE characters apart instead of the default 8.

*–tab1, tab2,..., tabn*

Set TAB characters at the columns specified by *tab1...*

**unexpand**

**–a** Insert TAB characters when replacing a run of two or more SPACE characters would produce a smaller output file.

## NAME

expr – evaluate arguments as a logical, arithmetic, or string expression

## SYNOPSIS

**expr** *argument...*

## DESCRIPTION

**expr** evaluates expressions as specified by its arguments. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument, so terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note: **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by '\'. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

*expr* \| *expr*
: Return the first *expr* if it is neither NULL nor **0**, otherwise returns the second *expr*.

*expr* \& *expr*
: Return the first *expr* if neither *expr* is NULL or **0**, otherwise returns **0**.

*expr* { =, \>, \>= , \<, \<=, **!=** } *expr*
: Return the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

*expr* { +, – } *expr*
: Addition or subtraction of integer-valued arguments.

*expr* { *, /, % } *expr*
: Multiplication, division, or remainder of the integer-valued arguments.

*string* : *regular-expression*
**match** *string regular-expression*
: The two forms of the matching operator above are synonymous. The matching operators : and **match** compare the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of **ed(1)**, except that all patterns are "anchored" (treated as if they begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

**substr** *string integer-1 integer-2*
: Extract the subtring of *string* starting at position *integer-1* and of length *integer-2* characters. If *integer-1* has a value greater than the length of *string*, **expr** returns a null string. If you try to extract more characters than there are in *string*, **expr** returns all the remaining characters from *string*. Beware of using negative values for either *integer-1* or *integer-2* as **expr** tends to run forever in these cases.

**index** *string character-list*
: Report the first position in *string* at which any one of the characters in *character-list* matches a character in *string*.

**length** *string*
: Return the length (that is, the number of characters) of *string*.

( **expr** ) Parentheses may be used for grouping.

## SYSTEM V DESCRIPTION

The operators **substr**, **index**, and **length** are not supported.

EXAMPLES

1. **a=‘expr $a + 1‘**

   Adds 1 to the shell variable **a**.

2. **# ’For $a equal to either "/usr/abc/file" or just "file"’**
   **expr $a : ’.*/\(.*\)’ \| $a**

   Returns the last segment of a path name (that is, the filename part). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. **# A better representation of example 2.**
   **expr //$a : ’.*/\(.*\)’**

   The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. **expr $VAR : ’.*’**

   Returns the number of characters in **$VAR**.

SEE ALSO

**ed**(1), **sh**(1), **test**(1V)

EXIT CODE

**expr** returns the following exit codes:

0 if the expression is neither null nor **0**

1 if the expression *is* null or **0**

2 for invalid expressions.

DIAGNOSTICS

**syntax error** for operator/operand errors

**non-numeric argument** if arithmetic is attempted on such a string

**division by zero** if an attempt to divide by zero is made

BUGS

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

**expr $a = ’=’**

looks like:

**expr = = =**

as the arguments are passed to **expr** (and they will all be taken as the = operator). The following works:

**expr X$a = X=**

Note: the **match**, **substr**, **length**, and **index** operators cannot themselves be used as ordinary strings. That is, the expression:

**example% expr index expurgatorious length**
**syntax error**
**example%**

generates the ‘**syntax error**’ message as shown instead of the value **1** as you might expect.

## NAME

file – determine the type of a file by examining its contents

## SYNOPSIS

**file** [ **–f** *ffile* ] [ **–cL** ] [ **–m** *mfile* ] *filename*...

## DESCRIPTION

**file** performs a series of tests on each *filename* in an attempt to determine what it contains. If the contents of a file appear to be ASCII text, **file** examines the first 512 bytes and tries to guess its language.

**file** uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type.

## OPTIONS

**–c** Check for format errors in the magic number file. For reasons of efficiency, this validation is not normally carried out. No file type-checking is done under **–c**.

**–f** *ffile* Get a list of filenames to identify from *ffile*.

**–L** If a file is a symbolic link, test the file the link references rather than the link itself.

**–m** *mfile*
Use *mfile* as the name of an alternate magic number file.

## EXAMPLE

This example illustrates the use of **file** on all the files in a specific user's directory:

```
example% pwd
/usr/blort/misc
example% file *
code:           mc68020 demand paged executable
code.c:         c program text
counts:         ascii text
doc:            roff, nroff, or eqn input text
empty.file:     empty
libz:           archive random library
memos:          directory
project:        symbolic link to /usr/project
script:         executable shell script
titles:         ascii text
s5.stuff:       cpio archive
example%
```

## FILES

**/etc/magic**

## SEE ALSO

**magic**(5)

## BUGS

**file** often makes mistakes. In particular, it often suggests that command files are C programs.

Does not recognize Pascal or LISP.

## NAME

find – find files by name, or by other characteristics

## SYNOPSIS

**find** *pathname-list expression*

## DESCRIPTION

**find** recursively descends the directory hierarchy for each pathname in the *pathname-list*, seeking files that match a boolean (logical) **expression** written using the operators listed below.

**find** does *not* follow symbolic links to other files or directories; it applies the selection criteria to the symbolic link itself, as if it were an ordinary file (see **ln**(1) for a description of symbolic links).

## USAGE

### Operators

In the descriptions, the argument *n* is used as a decimal integer where *+n* means more than *n*, *–n* means less than *n*, and *n* means exactly *n*.

**–fstype** *type* True if the filesystem to which the the file belongs is of type *type*, where *type* is typically **4.2** or **nfs**.

**–name** *filename* True if the *filename* argument matches the current file name. Shell argument syntax can be used if escaped (watch out for [, ? and *).

**–perm** *onum* True if the file permission flags exactly match the octal number *onum* (see **chmod**(1V)). If *onum* is prefixed by a minus sign, more flag bits (017777, see **chmod**(1V)) become significant and the flags are compared: *(flags&onum)==onum*.

**–prune** Always yields true. Has the side effect of pruning the search tree at the file. That is, if the current path name is a directory, **find** will not descend into that directory.

**–type** *c* True if the type of the file is *c*, where *c* is one of:

| | |
|---|---|
| **b** | for block special file **c** |
| **c** | for character special file |
| **d** | for directory |
| **f** | for plain file |
| **p** | for named pipe (FIFO) |
| **l** | for symbolic link |
| **s** | for socket |

**–links** *n* True if the file has *n* links.

**–user** *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** database, it is taken as a user ID.

**–nouser** True if the file belongs to a user *not* in the **/etc/passwd** database.

**–group** *gname* True if the file belongs to group *gname*. If *gname* is numeric and does not appear as a login name in the **/etc/group** database, it is taken as a group ID.

**–nogroup** True if the file belongs to a group *not* in the **/etc/group** database.

**–size** *n* True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.

**–inum** *n* True if the file has inode number *n*.

**–atime** *n* True if the file has been accessed in *n* days. Note: the access time of directories in *pathname-list* is changed by **find** itself.

**–mtime** *n* True if the file has been modified in *n* days.

**–ctime** *n* True if the file has been changed in *n* days. "Changed" means either that the file has been modified or some attribute of the file (its owner, its group, the number of links to it, etc.) has been changed.

**–exec** *command* True if the executed *command* returns a zero value as exit status. The end of *command* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current pathname.

**–ok** *command* Like **–exec** except that the generated command is written on the standard output, then the

standard input is read and the command executed only upon response **y**.

**–print** Always true; the current pathname is printed.

**–ls** Always true; prints current pathname together with its associated statistics. These include (respectively) inode number, size in kilobytes (1024 bytes), protection mode, number of hard links, user, group, size in bytes, and modification time. If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '–>'. The format is identical to that of **'ls–gilds'** ( See **ls**(1V)). Note: formatting is done internally, without executing the **ls** program.

**–cpio** *device* Always true; write the current file on *device* in **cpio**(5) format (5120-byte records).

**–ncpio** *device* Always true; write the current file on *device* in **'cpio–c'** format (5120-byte records).

**–newer** *file* True if the current file has been modified more recently than the argument *filename*.

**–xdev** Always true; find does *not* to traverse down into a file system different from the one on which current *argument* pathname resides.

**–depth** Always true; performs descent of the directory hierarchy so that all entries in a directory are acted on before the directory itself. This can be useful when **find** is used with **cpio**(1) to transfer files that are contained in directories without write permission.

**(** *expression* **)** True if the parenthesized *expression* is true (parentheses are special to the shell and must be escaped).

**!***primary* True if the *primary* is false (**!** is the unary *not* operator).

*primary1* [ **–a** ] *primary2*
True if both *primary1* and *primary2* are true. The **–a** is not required. It is implied by the juxtaposition of two primaries.

*primary1* **–o** *primary2*
True if either *primary1* or *primary2* is true (**–o** is the *or* operator).

**EXAMPLE**

In our local development system, we keep a file called TIMESTAMP in all the manual page directories. Here is how to find all entries that have been updated since TIMESTAMP was created:

```
example% find /usr/share/man/man2 –newer /usr/share/man/man2/TIMESTAMP –print
/usr/share/man/man2
/usr/share/man/man2/socket.2
/usr/share/man/man2/mmap.2
example%
```

To find all the files called **intro.ms** starting from the current directory:

```
example% find . –name intro.ms –print
./manuals/assembler/intro.ms
./manuals/sun.core/intro.ms
./manuals/driver.tut/intro.ms
./manuals/sys.manager/uucp.impl/intro.mss
./supplements/general.works/unix.introduction/intro.mss
./supplements/programming.tools/sccs/intro.mss
example%
```

To recursively print all files names in the current directory and below, but skipping SCCS directories:

```
example% find . –name SCCS –prune –o –print
example%
```

To recursively print all files names in the current directory and below, skipping the contents of SCCS directories, but printing out the SCCS directory name:

```
example% find . –print –name SCCS –prune
example%
```

To remove files beneath your home directory named **a.out** or ***.o** that have not been accessed for a week and that are not mounted using NFS:

```
example% cd
example% find . \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \; -o -fstype nfs -prune
```

FILES

**/etc/passwd**
**/etc/group**
**a.out**
***.o**

SEE ALSO

**cpio**(1), **sh**(1), **ln**(1), **chmod**(1V), **ls**(1V) **test**(1V), **cpio**(5), **fs**(5)

## NAME

finger – display information about users

## SYNOPSIS

**finger** [ *options* ] *name*...

## DESCRIPTION

By default, **finger** displays information about each logged-in user, including his or her: login name, full name, terminal name (prepended with a '*' if write-permission is denied), idle time, login time, and location (comment field in **/etc/ttytab** for users logged in locally, hostname for users logged in remotely) if known.

Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a **d** is present.

When one or more *name* arguments are given, more detailed information is given for each *name* specified, whether they are logged in or not. A *name* may be a first or last name, or an account name. Information is presented in a multi-line format, and includes, in addition to the information mentioned above:

the user's home directory and login shell
the time they logged in if they are currently logged in, or the time they last logged in if they are not, as well as the terminal or host from which they logged in and, if a terminal, the comment field in **/etc/ttytab** for that terminal
the last time they received mail, and the last time they read their mail
any plan contained in the file **.plan** in the user's home directory
and any project on which they are working described in the file **.project** (also in that directory)

If a *name* argument contains an at-sign, '@', then a connection is attempted to the machine named after the at-sign, and the remote finger daemon is queried. The data returned by that daemon is printed. If a long format printout is to be produced, the /W option is passed to the remote finger daemon.

## OPTIONS

**–m** Match arguments only on user name (not first or last name).

**–l** Force long output format.

**–s** Force short output format.

**–q** Force quick output format, which is similar to short format except that only the login name, terminal, and login time are printed.

**–i** Force "idle" output format, which is similar to short format except that only the login name, terminal, login time, and idle time are printed.

**–b** Suppress printing the user's home directory and shell in a long format printout.

**–f** Suppress printing the header that is normally printed in a non-long format printout.

**–w** Suppress printing the full name in a short format printout.

**–h** Suppress printing of the **.project** file in a long format printout.

**–p** Suppress printing of the **.plan** file in a long format printout.

## FILES

| | |
|---|---|
| **/etc/utmp** | who is logged in |
| **/etc/passwd** | for users' names |
| **/var/adm/lastlog** | last login times |
| **/etc/ttytab** | terminal locations |
| **/.plan** | plans |
| **/.project** | projects |

## SEE ALSO

**passwd**(1), **w**(1), **who**(1), **whois**(1)

**BUGS**

Only the first line of the **.project** file is printed.

## NAME

fmt, fmt_mail – simple text and mail-message formatters

## SYNOPSIS

**fmt** [ **–cs** ] [ *–width* ] [ *inputfile*... ]

**fmt_mail** [ **–cs** ] [ *–width* ] [ *inputfile* ... ]

## DESCRIPTION

**fmt** is a simple text formatter that fills and joins lines to produce output lines of (up to) the number of characters specified in the *–width* option. The default *width* is 72. **fmt** concatenates the *inputfile*s listed as arguments. If none are given, **fmt** formats text from the standard input.

Blank lines are preserved in the output, as is the spacing between words. **fmt** does not fill lines beginning with '.', for compatibility with **nroff**(1). Nor does it fill lines starting with 'From:' (but for full compatibility with **mail**(1), use **fmt_mail**).

Indention is preserved in the output, and input lines with differing indention are not joined (unless **–c** is used).

**fmt** can also be used as an in-line text filter for **vi**(1); the **vi** command:

**!}fmt**

reformats the text between the cursor location and the end of the paragraph.

**fmt_mail** is a script that formats and sends mail messages. It leaves mail header lines untouched, and runs the remainder of the message through **fmt -s**. The resulting message is passed along to **sendmail**(8), which routes it to the recipient.

## OPTIONS

**–c** Crown margin mode. Preserve the indention of the first two lines within a paragraph, and align the left margin of each subsequent line with that of the second line. This is useful for tagged paragraphs.

**–s** Split lines only. Do not join short lines to form longer ones. This prevents sample lines of code, and other such "formatted" text, from being unduly combined.

*–width* Fill output lines to up to *width* columns.

## SEE ALSO

**mail**(1), **nroff**(1), **vi**(1)

**NAME**

fold – fold long lines for display on an output device of a given width

**SYNOPSIS**

**fold** [ *–width* ] [ file ]

**DESCRIPTION**

Fold the contents of the specified *files*, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using **expand**(1) before using *fold*.

**SEE ALSO**

**expand**(1)

**BUGS**

Folding may not work correctly if underlining is present.

NAME

fontedit – a vfont screen-font editor

SYNOPSIS

**fontedit** [ *generic-tool-argument* ] ... [ *font_name* ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**fontedit** is an editor for fixed-width fonts in *vfont* format whose characters are no taller than 24 pixels (larger characters will not fit completely onto the screen). For a description of *vfont* format, see **vfont**(5).

OPTIONS

*generic-tool-argument*

**fontedit** accepts any generic tool argument as described in **sunview**(1). Otherwise, you can manipulate the tool using the Frame Menu.

COMMANDS

To edit a font, type '**fontedit**'. A *font_name* may be supplied on the command line or may be typed into the Control panel once the program has started. If it exists, the *font_name* file must be in *vfont* format. When the program starts, it displays a single large window containing four subwindows. From top to bottom, the four subwindows are:

1) The top subwindow, a message subwindow, displays messages, prompts, and warnings.

2) The second subwindow from the top, an Control panel, allows you to set global parameters for the entire font and specify operations for editing any single character. The options are:

**(Load)** Load in the font specified in the file name field. The program will warn you if you try to read over a modified font.

**(Store)** Store the current font onto disk with the name in file name field.

**(Quit)** Quit the program; warns you if you have modified the font.

**Font name:**

The name of the font.

**Max Width** and **Max Height:**

The size, in pixels, of the largest character in the font. If you edit an existing font, these parameters are set automatically; you must set them if you are creating a new font. Changing either of these values for an existing font may alter the glyph of some characters of the font. If the glyph size of a character is larger than the new max size, then that character is clipped to the new size (its bottom and right edges are moved in). However, if a glyph's size is smaller than the new size, the glyph is left alone.

**Caps Height** and **X-Height:**

The distance, in pixels, between the top of a capital and lowercase letter and the baseline. When an existing font is edited, the values of **Caps Height** and **X-Height** are estimated by *fontedit*, and may require some adjustment.

**Baseline:** The number of pixels from the top (that is, the upper left corner) of the character to the baseline. For an existing font, the value of the largest baseline distance is used. For a new font, each character will have the same baseline distance. If this value is changed, then the baseline distance for all characters in the font will be the new value.

**(Apply)** Apply the current values of **Max Width**, **Max Height**, **Caps Height**, **X-Height**, and **Baseline** to the font. That is, changes made to these values do not take effect until **Apply** is selected.

**Operation:**

This is a list of drawing and editing operations that you can perform on a character. For drawing, the left mouse button draws in black, and the middle draws in white. Operations are:

**Single Pt** Press a mouse button down and a grey cell will appear; move the mouse and the cell will follow it. Releasing the the button will draw.

**Pt Wipe** Pressing a button down will draw and moving with the button down will continue drawing until the button is released.

**Line** Button down marks the end point of a line; moving with the button down rubber bands a line; releasing button draws the line.

**Rect** Like **Line** except draws a rectangle.

**Cut** Button down marks one end of rectangle, and moving rubber bands the outline of the rectangle. Button up places the contents of the rectangle into a buffer and then "cuts" (draws in white) the rectangular region from the character. The **Paste** operation (below) gets the data from the buffer.

**Copy** Like **Cut** except that the region is just copied; no change is made to the character.

**Paste** Button down displays a rectangle the size of the region in the buffer. Moving with the button down moves the rectangle. Button up pastes the contents of the buffer into the character.
The contents of the paste buffer cannot be transferred between tools.
In **Copy** or **Cut** mode, holding down the shift key while pressing the left or middle mouse button will perform a **Paste** action. For best results, after placing a region in the buffer, press down the shift key and hold it down, then press down the mouse button. Release the mouse key to paste the region and then release the shift key.

3) The third subwindow echoes the characters in the current font as they are typed. Note that the cursor must be in this window in order to see the characters. Your character delete key will delete the echoed characters.

4) The bottom subwindow, the editing subwindow, displays eight smaller squares at its top; these are called **edit buttons**. The top section of each of these buttons contains a line of text in the form *nnn: c*, where *nnn* is the hexadecimal number of the character and *c* is the standard ASCII character corresponding to that number. In the lower section of the button the character of the current font, if it exists, is displayed. Clicking once over an editing button selects its character for editing.

   Just below this row of buttons is a box with the characters "0 9 A Z a z" in it. This box is called a **slider**. The slider allows you to scroll around in the font and select which section of the font you want displayed in the edit buttons. The black rectangle near "a" is an indicator which shows the section of the font that is displayed in the buttons above. To move the indicator, select it by pressing the left or middle mouse button down over the indicator and then move the mouse to the left or right with the button down; the indicator will slide along with the cursor. Releasing the button selects the new section of the font. A faster method of moving about in the font is to just press down and release the mouse button above the area you want without bothering to drag the indicator. Another method of scrolling through the characters of the font is to press a key on the keyboard when the cursor is in the bottom window; that character is the first one displayed in the edit buttons.

EDITING CHARACTERS:

To edit a character, click once over the edit button where the character is displayed. When you do this, an edit pad will appear in the bottom subwindow.

The edit pad consists of an editing area bordered by scales, a proof area, and 3 command buttons. The editing area is **Max Width** by **Max Height** when the pad opens, and displays a magnified view of the selected character. Black squares indicate foreground pixels. The editing area is surrounded by scales which show the current **Caps Height, X-Height** and **Baseline** in reverse video.

Just outside the scales, on the top, right side, and bottom of the pad, are three small boxes with the capital letters "R", "B", and "A" in them. These boxes are movable sliders that change the right edge, bottom edge, and x-axis advance of the character respectively. In a fixed-width font, these values are usually the same for all characters; however, in a variable-width font these controls can be used to set these properties for each character.

To the right of the pad is the proof area where the character is displayed at normal (that is, screen) resolution and three buttons. The three buttons are:

**Undo** Clicking the left or middle mouse button undoes the last operation.

**Store** Stores the current representation of the character in the font.

**Quit** Closes the edit pad.

In the bottom subwindow, the right mouse button displays a menu of operations. These operations are the same as those in the control panel discussed above; you can select the current operation by either picking the operation in the control panel or by selecting the appropriate menu with the right button of the mouse. When the cursor is in the other subwindows, the right button displays the standard tool menu.

## FILES

**/usr/lib/fonts/fixedwidthfonts**
Sun-supplied screen fonts

## SEE ALSO

**sunview**(1), **vswap**(1), **vfont**(5)

## BUGS

Results are unpredictable with variable-width fonts. The baseline should be greater than 0 or else the font cannot be read in by **fontedit** or by **sunview**(1).

NAME

foption – determine available floating-point code generation options

SYNOPSIS

**foption** [ –*ftype* ]

DESCRIPTION

**foption** has two uses on Sun-2 and Sun-3 systems. Its action is undefined on Sun-4 systems since there are no floating-point code generation options.

Called without an argument, it sends a string to standard output which is the compiler floating-point option corresponding to the type of floating-point hardware that would be used by a program compiled with **–fswitch**. Exit status is undefined. This usage is intended for interactively determining available floating-point hardware. On Sun-2 or Sun-3 systems without floating-point hardware, the result would be

```
example% foption
soft
```

corresponding to the compiler option **–fsoft**.

Called with an argument which is one of the compiler floating-point options **–ffpa**, **–f68881**, **–fsoft**, or **–fswitch**, it produces no output but returns exit status 0 (true) if a program compiled with that option could execute on this machine, and status 1 (false) otherwise. Thus **foption –fsoft** and **foption –fswitch** always produce exit status 0. This usage is intended for shell scripts and Makefiles that, for instance, select different executable files or link with different libraries according to the floating-point hardware present.

OPTIONS

–*ftype* Return exit status 0 if a program compiled –*ftype* could execute on this machine.

SEE ALSO

cc(1V), **fpaversion**(8), **mc68881version**(8)

**NAME**

from – display the sender and date of newly-arrived mail messages

**SYNOPSIS**

**from** [ **–s** *sender* ] [ *username* ]

**DESCRIPTION**

*From*name prints out the mail header lines in your mailbox file to show you who your mail is from. If *username* is specified, then *username*'s mailbox is examined instead of your own.

**OPTIONS**

**–s***sender*

Only display headers for mail sent by *sender*.

**FILES**

**/var/spool/mail/***

**SEE ALSO**

biff(1), mail(1), prmail(1)

NAME

ftp – file transfer program

SYNOPSIS

**ftp** [ **–dgintv** ] [ *hostname* ]

AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**ftp** is the user interface to the ARPANET standard File Transfer Protocol (FTP). **ftp** transfers files to and from a remote network site.

The client host with which **ftp** is to communicate may be specified on the command line. If this is done, **ftp** immediately attempts to establish a connection to an FTP server on that host; otherwise, **ftp** enters its command interpreter and awaits instructions from the user. When **ftp** is awaiting commands from the user, it displays the prompt '**ftp>**'.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

**–d** Enable debugging.

**–g** Disable filename "globbing."

**–i** Turn off interactive prompting during multiple file transfers.

**–n** Do not attempt "auto-login" upon initial connection. If auto-login is enabled, **ftp** checks the **.netrc** file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, **ftp** will prompt for the login name of the account on the remote machine (the default is the login name on the local machine), and, if necessary, prompts for a password and an account with which to login.

**–t** Enable packet tracing (unimplemented).

**–v** Show all responses from the remote server, as well as report on data transfer statistics. This is turned on by default if **ftp** is running interactively with its input coming from the user's terminal.

COMMANDS

**!** [ *command* ]

Run *command* as a shell command on the local machine. If no *command* is given, invoke an interactive shell.

**$** *macro-name* [ *args* ]

Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

**account** [ *passwd* ]

Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

**append** *local-file* [ *remote-file* ]

Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any **ntrans** or **nmap** setting. File transfer uses the current settings for "representation type", "file structure", and "transfer mode".

**ascii** Set the "representation type" to "network ASCII". This is the default type.

**bell** Sound a bell after each file transfer command is completed.

**binary** Set the "representation type" to "image".

**bye** Terminate the FTP session with the remote server and exit **ftp**. An EOF will also terminate the

session and exit.

**case** Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

**cd** *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

**cdup** Change the remote machine working directory to the parent of the current remote machine working directory.

**close** Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

**cr** Toggle RETURN stripping during "network ASCII" type file retrieval. Records are denoted by a RETURN/LINEFEED sequence during "network ASCII" type file transfer. When **cr** is on (the default), RETURN characters are stripped from this sequence to conform with the UNIX system single LINEFEED record delimiter. Records on non-UNIX-system remote hosts may contain single LINEFEED characters; when an "network ASCII" type transfer is made, these LINEFEED characters may be distinguished from a record delimiter only when **cr** is off.

**delete** *remote-file*
Delete the file *remote-file* on the remote machine.

**debug** [ *debug-value* ]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, **ftp** prints each command sent to the remote machine, preceded by the string '**-->**'.

**dir** [ *remote-directory* ] [ *local-file* ]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is '–', output is sent to the terminal.

**disconnect**
A synonym for **close**.

**form** [ *format-name* ]
Set the carriage control format subtype of the "representation type" to *format-name*. The only valid *format-name* is **non-print**, which corresponds to the default "non-print" subtype.

**get** *remote-file* [ *local-file* ]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current **case**, **ntrans**, and **nmap** settings. The current settings for "representation type", "file structure", and "transfer mode" are used while transferring the file.

**glob** Toggle filename expansion, or "globbing", for **mdelete**, **mget** and **mput**. If globbing is turned off, filenames are taken literally.

Globbing for **mput** is done as in **csh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine, and the lists are not merged.

Expansion of a directory name is likely to be radically different from expansion of the name of an ordinary file: the exact result depends on the remote operating system and FTP server, and can be previewed by doing '**mls** *remote-files* –'.

**mget** and **mput** are not meant to transfer entire directory subtrees of files. You can do this by transferring a **tar**(1) archive of the subtree (using a "representation type" of "image" as set by the **binary** command).

**hash** Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 1024

bytes.

**help** [ *command* ]
Print an informative message about the meaning of *command*. If no argument is given, **ftp** prints a list of the known commands.

**lcd** [ *directory* ]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

**ls** [ *remote-directory* ] [ *local-file* ]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is '–', the output is sent to the terminal.

**macdef** *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive NEWLINE characters in a file or RETURN characters from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed.

The macro processor interprets '$' and '\' as special characters. A '$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '$' followed by an 'i' signals that macro processor that the executing macro is to be looped. On the first pass '$i' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '$'.

**mdelete** [ *remote-files* ]
Delete the *remote-files* on the remote machine.

**mdir** *remote-files local-file*
Like **dir**, except multiple remote files may be specified. If interactive prompting is on, **ftp** will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

**mget** *remote-files*
Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to **case**, **ntrans**, and **nmap** settings. Files are transferred into the local working directory, which can be changed with '**lcd** *directory*'; new local directories can be created with '**! mkdir** *directory*'.

**mkdir** *directory-name*
Make a directory on the remote machine.

**mls** *remote-files local-file*
Like **ls**(1V), except multiple remote files may be specified. If interactive prompting is on, **ftp** will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

**mode** [ *mode-name* ]
Set the "transfer mode" to *mode-name*. The only valid *mode-name* is **stream**, which corresponds to the default "stream" mode.

**mput** *local-files*
Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to **ntrans** and **nmap** settings.

**nmap** [ *inpattern outpattern* ]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences **$1**, **$2**, ..., **$9** in *inpattern*. Use \ to prevent this special treatment of the **$** character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values.

For example, given *inpattern* **$1.$2** and the remote file name **mydata.data**, **$1** would have the value "mydata", and **$2** would have the value "data".

The *outpattern* determines the resulting mapped filename. The sequences **$1**, **$2**, ..., **$9** are replaced by any value resulting from the *inpattern* template. The sequence **$0** is replaced by the original filename. Additionally, the sequence '[*seq1* , *seq2* ]' is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*.

For example, the command '**nmap $1.$2.$3 [$1,$2].[$2,file]**' would yield the output filename **myfile.data** for input filenames **myfile.data** and **myfile.data.old**, **myfile.file** for the input filename **myfile**, and **myfile.myfile** for the input filename **.myfile**. SPACE characters may be included in *outpattern*, as in the example '**nmap $1 | sed "s/ *$//" > $1**'. Use the \ character to prevent special treatment of the '$', '[', ']' and ',' characters.

**ntrans** [ *inchars* [ *outchars* ] ]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename, and characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

**open** *host* [ *port* ]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, **ftp** will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), **ftp** will also attempt to automatically log the user in to the FTP server (see below).

**prompt** Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. By default, prompting is turned on. If prompting is turned off, any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

**proxy** *ftp-command*

Execute an FTP command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command '**proxy ?**' to see other FTP commands executable on the secondary connection.

The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control

connection to the host on the primary control connection.

Third party file transfers depend upon support of the **PASV** command by the server on the secondary control connection.

**put** *local-file* [ *remote-file* ]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any **ntrans** or **nmap** settings in naming the remote file. File transfer uses the current settings for "representation type", "file structure", and "transfer mode".

**pwd** Print the name of the current working directory on the remote machine.

**quit** A synonym for **bye**.

**quote** *arg1 arg2 ...*
Send the arguments specified, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

**recv** *remote-file* [ *local-file* ]
A synonym for **get**.

**remotehelp** [ *command-name* ]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

**rename** *from to*
Rename the file *from* on the remote machine to have the name *to*.

**reset** Clear reply queue. This command re-synchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

**rmdir** *directory-name*
Delete a directory on the remote machine.

**runique**
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a **'.1'** is appended to the name. If the resulting name matches another existing file, a **'.2'** is appended to the original name. If this process continues up to **'.99'**, an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note: **runique** will not affect local files generated from a shell command (see below). The default value is off.

**send** *local-file* [ *remote-file* ]
A synonym for **put**.

**sendport**
Toggle the use of **PORT** commands. By default, **ftp** will attempt to use a **PORT** command when establishing a connection for each data transfer. The use of **PORT** commands can prevent delays when performing multiple file transfers. If the **PORT** command fails, **ftp** will use the default data port. When the use of **PORT** commands is disabled, no attempt will be made to use **PORT** commands for each data transfer. This is useful when connected to certain FTP implementations that ignore **PORT** commands but incorrectly indicate they have been accepted.

**status** Show the current status of **ftp**.

**struct** [ *struct-name* ]
Set the "file structure" to *struct-name*. The only valid *struct-name* is **file**, which corresponds to the default "file" structure.

**sunique**
Toggle storing of files on remote machine under unique file names. The remote FTP server must support the **STOU** command for successful completion. The remote server will report the unique

name. Default value is off.

**tenex** Set the "representation type" to that needed to talk to TENEX machines.

**trace** Toggle packet tracing (unimplemented).

**type** [ *type-name* ]
Set the "representation type" to *type-name*. The valid *type-name*s are **ascii** for "network ASCII", **binary** or **image** for "image", and **tenex** for "local byte size" with a byte size of 8 (used to talk to TENEX machines). If no type is specified, the current type is printed. The default type is "network ASCII".

**user** *user-name* [ *password* ] [ *account* ]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, **ftp** will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless **ftp** is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

**verbose** Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose mode is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose mode is on if **ftp**'s commands are coming from a terminal, and off otherwise.

**?** [ *command* ]
A synonym for **help.**

Command arguments which have embedded spaces may be quoted with quote (") marks.

If any command argument which is not indicated as being optional is not specified, **ftp** will prompt for that argument.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually CTRL-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol **ABOR** command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for **ABOR** processing. If the remote server does not support the **ABOR** command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when **ftp** has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the **ABOR** processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local **ftp** program must be killed by hand.

FILE NAMING CONVENTIONS

Local files specified as arguments to **ftp** commands are processed according to the following rules.

1) If the file name '–' is specified, the standard input (for reading) or standard output (for writing) is used.

2) If the first character of the file name is '|', the remainder of the argument is interpreted as a shell command. **ftp** then forks a shell, using **popen**(3S) with the argument supplied, and reads (writes) from the standard output (standard input) of that shell. If the shell command includes SPACE characters, the argument must be quoted; for example '"**| ls -lt**"'. A particularly useful example of this mechanism is: '**dir | more**'.

3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the **csh**(1); see the **glob** command. If the **ftp** command expects a single local file (for example, **put**), only the first filename generated by the "globbing" operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the

remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer.

The "representation type" may be one of "network ASCII", "EBCDIC", "image", or "local byte size" with a specified byte size (for PDP-10's and PDP-20's mostly). The "network ASCII" and "EBCDIC(rq types have a further subtype which specifies whether vertical format control (NEWLINE characters, form feeds, etc.) are to be passed through ("non-print"), provided in TELNET format ("TELNET format controls"), or provided in ASA (FORTRAN) ("carriage control (ASA)") format. **ftp** supports the "network ASCII" (subtype "non-print" only) and "image" types, plus "local byte size" with a byte size of 8 for communicating with TENEX machines.

The "file structure" may be one of "file" (no record structure), "record", or "page". **ftp** supports only the default value, which is "file".

The "transfer mode" may be one of "stream", "block", or "compressed". **ftp** supports only the default value, which is "stream".

SEE ALSO

**csh**(1), **ls**(1V), **rcp**(1C), **tar**(1), **popen**(3S), **netrc**(5), **ftpd**(8C)

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2 BSD code handling transfers with a "representation type" of "network ASCII" has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using a "representation type" of "network ASCII". Avoid this problem by using the "image" type.

**NAME**

gcore – get core images of running processes

**SYNOPSIS**

**gcore** [ **–o** *filename* ] *process-id* ...

**DESCRIPTION**

**gcore** creates a core image of each specified process. Such an image can be used with **adb**(1) or **dbx**(1). The name of the core image file for the process whose process ID is *process-id* will be **core.***process-id*.

**OPTIONS**

**–o** *filename*

Substitute *filename* in place of **core** as the first part of the name of the core image files.

**FILES**

**core.***process-id* core images

**SEE ALSO**

**kill**(1), **csh**(1), **adb**(1), **dbx**(1), **ptrace**(2)

## NAME

get – get a version of an SCCS file

## SYNOPSIS

**/usr/sccs/get** [ **–begkmnpst** ] [ **–l** [ **p** ] ] [ **–a** *seq-no.* ] [ **–c** *cutoff* ] [ **–G***newname* ] [ **–i** *list* ] [ **–r***SID* ] [ **–x** *list* ] *filename* ...

## DESCRIPTION

**get** generates an ASCII text file from each named SCCS file according to the specified option. Arguments may be specified in any order, options apply to all named SCCS files. If a directory is named, **get** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with 's.') and unreadable files are silently ignored. If a name of '–' is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the **g-file** whose name is derived from the SCCS file name by simply removing the leading 's.'; (see also FILES, below).

## OPTIONS

Options are explained below as though only one SCCS file is to be processed, but the effects of any option argument applies independently to each named file.

**–b** Used with the **–e** option to indicate that the new delta should have an SID in a new branch as shown in Table 1. This option is ignored if the **b** flag is not present in the file (see **admin**(1)) or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.)

Note: A branch delta may always be created from a non-leaf delta.

**–e** This **get** is for editing or making a change (delta) to the SCCS file with a subsequent use of **delta**. A '**/usr/sccs/get –e**' applied to a particular version (SID) of the SCCS file prevents further '**/usr/sccs/get –e**' commands on the same SID until **delta** is run or the **j** (joint edit) flag is set in the SCCS file (see **admin**(1)). Concurrent use of '**/usr/sccs/get –e**' for different SIDs is always allowed.

If the **g-file** generated by a '**/usr/sccs/get –e**' is accidentally ruined in the process of editing it, it may be regenerated by re-running a **get** with the **–k** option in place of the **–e** option.

SCCS file protection specified with the ceiling, floor, and authorized user list stored in the SCCS file (see **admin**(1)) are enforced when the **–e** option is used.

**–g** Do not actually retrieve text from the SCCS file. It is primarily used to generate an **l-file**, or to verify the existence of a particular SID.

**–k** Suppress replacement of identification keywords (see below) in the retrieved text by their value. The **–k** option is implied by the **–e** option.

**–m** Precede each text line retrieved from the SCCS file with the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal TAB, followed by the text line.

**–n** Precede each generated text line with the **%M%** identification keyword value (see below). The format is: **%M%** value, followed by a horizontal TAB followed by the text line. When both the **–m** and **–n** options are used, the format is: **%M%** value, followed by a horizontal TAB, followed by the **–m** option generated format.

**–p** Write the text retrieved from the SCCS file to the standard output. No **g-file** is created. All output which normally goes to the standard output goes to the standard error file instead, unless the **–s** option is used, in which case it disappears.

**–s** Suppress all output normally written on the standard output. However, fatal error messages (which always go to the standard error file) remain unaffected.

**–t** Access the most recently created (top) delta in a given release (for example, **–r1**), or release and level (for example, **–r1.2**).

**–l[p]** Write a delta summary into an **l-file**. If **–lp** is used, the delta summary is written on the standard output and the **l-file** is not created. See FILES for the format of the **l-file**.

**–a** *seq-no.*

The delta sequence number of the SCCS file delta (version) to be retrieved (see **sccsfile**(5)). This option is used by the **comb**(1) command; it is not a generally useful option, and users should not use it. If both the **–r** and **–a** options are specified, the **–a** option is used. Care should be taken when using the **–a** option in conjunction with the **–e** option, as the SID of the delta to be created may not be what one expects. The **–r** option can be used with the **–a** and **–e** options to control the naming of the SID of the delta to be created.

**–c** *cutoff*

*cutoff* date-time, in the form: YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **–c7502** is equivalent to **–c750228235959**. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: '**–c77/2/29:22:25**'. Note: this implies that one may use the **%E%** and **%U%** identification keywords.

**–G***newname*

If a **get** is allowed on *filename* (*filename* is not writable by anyone) place the version that **get** produces in a file called *newname*.

**–i** *list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<*list*> ::= <*range* > | <*list*> , <*range*>
<*range*> ::= SID | SID – SID

SID, the SCCS ID of a delta, may be in any form shown in the 'SID Specified' column of Table 1. Partial SIDs are interpreted as shown in the 'SID Retrieved' column of Table 1.

**–r** *SID* The SCCS ID string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by **delta**(1) if the **–e** option is also used), as a function of the SID specified.

**–x** *list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **–i** option for the *list* format.

For each file processed, **get** responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **–e** option is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each filename is printed (preceded by a NEWLINE) before it is processed. If the **–i** option is used included deltas are listed following the notation '**Included**'; if the **–x** option is used, excluded deltas are listed following the notation '**Excluded**'.

TABLE 1. Determination of SCCS Identification String

| SID* Specified | –b Option Used† | Other Conditions | SID Retrieved | SID of Delta to be Created |
|---|---|---|---|---|
| none‡ | no | R defaults to mR | mR.mL | mR.(mL+1) |
| none‡ | yes | R defaults to mR | mR.mL | mR.mL.(mB+1).1 |
| R | no | R > mR | mR.mL | R.1*** |
| R | no | R = mR | mR.mL | mR.(mL+1) |
| R | yes | R > mR | mR.mL | mR.mL.(mB+1).1 |
| R | yes | R = mR | mR.mL | mR.mL.(mB+1).1 |
| R | – | R < mR and R does *not* exist | hR.mL** | hR.mL.(mB+1).1 |
| R | – | Trunk succ.# in release > R and R exists | R.mL | R.mL.(mB+1).1 |
| R.L | no | No trunk succ. | R.L | R.(L+1) |
| R.L | yes | No trunk succ. | R.L | R.L.(mB+1).1 |
| R.L | – | Trunk succ. in release ≥ R | R.L | R.L.(mB+1).1 |
| R.L.B | no | No branch succ. | R.L.B.mS | R.L.B.(mS+1) |
| R.L.B | yes | No branch succ. | R.L.B.mS | R.L.(mB+1).1 |
| R.L.B.S | no | No branch succ. | R.L.B.S | R.L.B.(S+1) |
| R.L.B.S | yes | No branch succ. | R.L.B.S | R.L.(mB+1).1 |
| R.L.B.S | – | Branch succ. | R.L.B.S | R.L.(mB+1).1 |

* 'R', 'L', 'B', and 'S' are the 'release', 'level', 'branch', and 'sequence' components of the SID, respectively; 'm' means 'maximum'. Thus, for example, 'R.mL' means 'the maximum level number within release R'; 'R.L.(mB+1).1' means 'the first sequence number on the *new* branch (that is, maximum branch number plus one) of level L within release R'. Note: if the .SM SID specified is of the form 'R.L', 'R.L.B', or 'R.L.B.S', each of the specified components *must* exist.

** 'hR' is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** Forces creation of the *first* delta in a *new* release.

\# Successor.

† The **–b** option is effective only if the **b** flag (see **admin**(1)) is present in the file. An entry of '–' means 'irrelevant'.

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

**IDENTIFICATION KEYWORDS**

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

| *Keyword* | *Value* |
|---|---|
| **%M%** | Module name: either the value of the **m** flag in the file (see **admin**(1)), or if absent, the name of the SCCS file with the leading **s.** removed. |
| **%I%** | SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text. |
| **%R%** | Release. |
| **%L%** | Level. |

| | |
|---|---|
| **%B%** | Branch. |
| **%S%** | Sequence. |
| **%D%** | Current date (YY/MM/DD). |
| **%H%** | Current date (MM/DD/YY). |
| **%T%** | Current time (HH:MM:SS). |
| **%E%** | Date newest applied delta was created (YY/MM/DD). |
| **%G%** | Date newest applied delta was created (MM/DD/YY). |
| **%U%** | Time newest applied delta was created (HH:MM:SS). |
| **%Y%** | Module type: value of the **t** flag in the SCCS file (see **admin**(1)). |
| **%F%** | SCCS file name. |
| **%P%** | Fully qualified SCCS file name. |
| **%Q%** | The value of the **q** flag in the file (see **admin**(1)). |
| **%C%** | Current line number. This keyword is intended for identifying messages output by the program such as **'this shouldn't have happened'** type errors. It is *not* intended to be used on every line to provide sequence numbers. |
| **%Z%** | The 4-character string @(#) recognizable by **what**(1). |
| **%W%** | A shorthand notation for constructing **what** strings for program files.<br>**%W%** = **%Z%%M%**<*horizontal*TAB>**%I%** |
| **%A%** | Another shorthand notation for constructing **what** strings.<br>**%A%** = **%Z%%Y% %M% %I%%Z%** |

**FILES**

Several auxiliary files may be created by **get**, These files are known generically as the **g-file, l-file, p-file,** and **z-file**. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The **g-file** is an exception to this scheme: the **g-file** is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c, l.xyz.c, p.xyz.c**, and **z.xyz.c**, respectively.

The **g-file**, which contains the generated text, is created in the current directory (unless the **–p** option is used). A **g-file** is created in all cases, whether or not any lines of text were generated by the **get**. It is owned by the real user. If the **–k** option is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The **l-file** contains a table showing which deltas were applied in generating the retrieved text. The **l-file** is created in the current directory if the **–l** option is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the **l-file** have the following format:

a. A blank character if the delta was applied; * otherwise.
b. A blank character if the delta was applied or was not applied and ignored; * if the delta was not applied and was not ignored.
c. A code indicating a "special" reason why the delta was or was not applied:
   'I': Included.
   'X': Excluded.
   'C': Cut off (by a **–c** option).
d. Blank.
e. SCCS identification (SID).
f. TAB character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created the delta.

The comments and MR data follow on subsequent lines, indented one horizontal TAB character. A blank line terminates each entry.

The **p-file** passes information resulting from a '**/usr/sccs/get** **–e**' along to **delta**. Its contents are also used to prevent a subsequent execution of a '**/usr/sccs/get** **–e**' for the same SID until **delta** is executed or the joint edit flag, **j**, (see **admin**(1)) is set in the SCCS file. The **p-file** is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the **p-file** is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the **get** was executed, followed by a blank and the **–i** option if it was present, followed by a blank and the **–x** option if it was present, followed by a NEW-LINE. There can be an arbitrary number of lines in the **p-file** at any time; no two lines can have the same new delta SID.

The **z-file** serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (that is, **get**) that created it. The **z-file** is created in the directory containing the SCCS file for the duration of **get**. The same protection restrictions as those for the **p-file** apply for the **z-file**. The **z-file** is created mode 444.

## SEE ALSO

**sccs**(1), **admin**(1), **delta**(1), **help**(1), **prs**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use **help** for explanations.

## BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, only one file may be named when the **–e** option is used.

NAME

get_selection – copy the contents of a SunView selection to the standard output

SYNOPSIS

**get_selection** [ *rank* ] [ **t** *seconds* ] [ **D** ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**get_selection** prints the contents of the indicated selection on standard out. A selection is a collection of objects (for instance, characters) picked with the mouse in the SunView window system.

OPTIONS

*rank* Indicate which selection is to be printed:
**1**: primary;
**2**: secondary;
**3**: clipboard.

The default is primary.

**t** *seconds*
Indicate how many seconds to wait for the holder of a selection to respond to a request before giving up. The default is 6 seconds.

**D** Debugging. Inquire through a special debugging service for the selection, rather than accessing the standard service. Useful only for debugging window applications which are clients of the selection library.

EXAMPLE

The following line in a SunView root menu file provides a menu command to print the primary selection on the user's default printer:

```
"Print It"  csh -c get_selection | lpr
```

SEE ALSO

*SunView 1 Beginner's Guide*

NAME

getopt – parse command options in shell scripts

SYNOPSIS

**set – – 'getopt** *opstring* **$*'**
**set argv = ('getopt** *opstring* **$*')**

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**getopt** is used to break up options in command lines for easy parsing by shell scripts, and to check for legal options. *optstring* is a string of option letters to recognize, (see **getopt**(3)). If a letter is followed by a colon, the option is expected to have an argument — which may or may not be separated by white space.

(The '– –' following **set** indicates that the Bourne shell is to pass arguments beginning with a dash as parameters to the script.)

If '–' appears on the command line that invokes the script, **getopt** uses it to delimit the end of options it is to parse (see example below). If used explicitly, **getopt** will recognize it; otherwise, **getopt** will generate it at the first argument it encounters that has no '–'. In either case, **getopt** places it at the end of the options. The positional parameters (**$1 $2**...) of the shell are reset so that each option in *optstring* is broken out and preceded by a '–', along with the argument (if any) for each.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
#! /usr/bin/sh
set -- getopt abo: $*'
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a| -b) FLAG =$i; shift;;
        -o)             OARG =$2; shift 2;;
        --)             shift; break;;
        esac
done
```

This code will accept any of the following command lines as equivalent:

```
cmd -a -o arg  f1  f2
cmd -aoarg  f1  f2
cmd -oarg -a  f1  f2
cmd -a -oarg --  f1  f2
```

SEE ALSO

**csh**(1), **getopts**(1), **sh**(1), **getopt**(3)

DIAGNOSTICS

**getopt** prints an error message on the standard error when it encounters an option letter not included in *optstring*.

**NOTES**

**getopts**(1) is preferred.

NAME

getopts, getoptcvt – parse command options in shell scripts

SYNOPSIS

**getopts** *optstring name* [ *argument*... ]

**getoptcvt** [ **–b** ] *filename*

DESCRIPTION

**getopts** is used by shell procedures to parse positional parameters and to check for legal options. It should be used in place of the **getopt**(1) command. It supports the following command syntax rules:

- Option names must be one character long.
- All options must be preceded by '–'.
- Options with no arguments may be grouped after a single '–'.
- The first option-argument following an option must be preceded by white space.
- Option-arguments cannot be optional.
- Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (that is, '**–o xxx,z,yy**' or '**–o "xxx z yy"**').
- All options must precede operands on the command line.
- '**––**' may be used to indicate the end of the options.

*optstring* must contain the option letters the command using **getopts** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, **getopts** will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, **getopts** places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, **getopts** exits with a non-zero exit status. The special option '**––**' may be used to delimit the end of the options.

By default, **getopts** parses the positional parameters. If extra arguments (*argument* ...) are given on the **getopts** command line, **getopts** will parse them instead.

**getoptcvt** reads the shell script in *filename*, converts it to use **getopts** instead of **getopt**, and writes the results on the standard output.

OPTIONS

**getoptcvt**

**–b** Generate a script that will be portable to earlier releases of the UNIX system. The script will determine at run time whether to invoke **getopts** or **getopt**.

**EXAMPLE**

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
done
shift `expr $OPTIND - 1`
```

This code will accept any of the following as equivalent:

**cmd** –**a** –**b** –**o** "**xxx z yy**" *filename*
**cmd** –**a** –**b** –**o** "**xxx z yy**" — *filename*
**cmd** –**ab** –**o** **xxx,z,yy** *filename*
**cmd** –**ab** –**o** "**xxx z yy**" *filename*
**cmd** –**o** **xxx,z,yy** –**b** –**a** *filename*

**SEE ALSO**

**getopt**(1), **sh**(1), **getopt**(3)

**WARNING**

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

**DIAGNOSTICS**

**getopts** prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

gfxtool – run graphics programs in a SunView window

SYNOPSIS

**gfxtool** [ **–C** ] [ *program* [ *arguments* ] ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**gfxtool** is a standard tool provided with the *SunView* environment. It allows you to run graphics programs that don't overwrite the terminal emulator from which they run.

**gfxtool** has two subwindows: a terminal subwindow and an empty subwindow. The terminal subwindow contains a running shell, just like the shelltool (see **shelltool**(1)). Programs invoked in the terminal subwindow can run in the empty subwindow. You can move the boundary between these two subwindows as described in **sunview**(1). If you wish, you can make **gfxtool** your console by entering a first argument of -C.

Normally you can use the mouse and keyboard anywhere in the empty subwindow to access frame functions. However, some graphics programs which run in this window may take over inputs directed to it. For example, SunCore uses the mouse and keyboard for its own input. When you run such tools, access the Frame Menu from the tool boundaries or frame header.

OPTIONS

**–C** Redirect system console output to this instance of **gfxtool.**

**gfxtool** also accepts all of the generic tool arguments; see **sunview**(1) for a list of these arguments.

If a *program* argument is present, **gfxtool** runs it. If there are no arguments, **gfxtool** runs the program corresponding to your SHELL environment variable. If this environment variable is not available, then **gfxtool** runs **/usr/bin/sh.**

FILES

**/.ttyswrc**
**/usr/bin/gfxtool**
**/usr/demo/***

SEE ALSO

**shelltool**(1), **sunview**(1), **gp_demos**(6)

BUGS

If more than 256 characters are input to a terminal emulator subwindow without an intervening NEWLINE, the terminal emulator may hang. If this occurs, display the Frame Menu; the **'TTY Hung?'** submenu there has one item, **'Flush input'**, that you can invoke to correct the problem.

NAME

gprof – display call-graph profile data

SYNOPSIS

**gprof** [ **–abcsz** ] [ **–e** *filename* ] [ **–E** *filename* ] [ **–f** *filename* ] [ **–F** *filename* ]
[ *image-file* [ *profile-file* ... ] ]

DESCRIPTION

**gprof** produces an execution profile of a program. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file which is created by programs compiled with the **–pg** option of **cc**(1V) and other compilers. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the executable image file *image-file* (**a.out** by default) is read and correlated with the call graph profile file *profile-file* (**gmon.out** by default). If more than one profile file is specified, the **gprof** output shows the sum of the profile information in the given profile files.

First, execution times for each routines are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. The first listing shows the functions sorted according to the time they represent, including the time of their call graph descendants. Below each function entry is shown its (direct) call-graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendants is propagated to its (direct) call-graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

Next, a flat profile is given, similar to that provided by **prof**(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time. Finally, an index showing the correspondence between function names and call-graph profile index numbers.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(3)).

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call-graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

The profiled program must call **exit**(2) or return normally for the profiling information to be saved in the **gmon.out** file.

OPTIONS

**–a** Suppress printing statically declared functions. If this option is given, all relevant information about the static function (for instance, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the **a.out** file.

**–b** Brief. Suppress descriptions of each field in the profile.

**–c** The static call-graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

**–s** Produce a profile file **gmon.sum** which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of **gprof** (probably also with a **–s**) option to accumulate profile data across several runs of an **a.out** file.

**–z** Display routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **–c** option for discovering which routines were never called.

**–e** *filename*
Suppress printing the graph profile entry for routine *filename* and all its descendants (unless they have other ancestors that are not suppressed). More than one **–e** option may be given. Only one *filename* may be given with each **–e** option.

**–E** *filename*

Suppress printing the graph profile entry for routine *filename* (and its descendants) as **–e**, above, and also exclude the time spent in *filename* (and its descendants) from the total and percentage time computations. More than one **–E** option may be given. For example:

'**–E** *mcount* **–E** *mcleanup*'

is the default.

**–f** *filename*

Print the graph profile entry only for routine *filename* and its descendants. More than one **–f** option may be given. Only one *filename* may be given with each **–f** option.

**–F** *filename*

Print the graph profile entry only for routine *filename* and its descendants (as **–f**, above) and also use only the times of the printed routines in total time and percentage computations. More than one **–F** option may be given. Only one *filename* may be given with each **–F** option. The **–F** option overrides the **–E** option.

## ENVIRONMENT

**PROFDIR**

If this environment variable contains a value, place profiling output within that directory, in a file named *pid.programname*. *pid* is the process ID, and *programname* is the name of the program being profiled, as determined by removing any path prefix from the **argv[0]** with which the program was called. If the variable contains a NULL value, no profiling output is produced. Otherwise, profiling output is placed in the file **gmon.out**.

## FILES

| | |
|---|---|
| **a.out** | executable file containing namelist |
| **gmon.out** | dynamic call-graph and profile |
| **gmon.sum** | summarized dynamic call-graph and profile |
| **$PROFDIR**/*pid.programname* | |

## SEE ALSO

**cc**(1V), **prof**(1), **tcov**(1), **exit**(2), **profil**(2), **monitor**(3), **prof**(3)

Graham, S.L., Kessler, P.B., McKusick, M.K., '*gprof: A Call Graph Execution Profiler*', *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

## BUGS

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call-graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

## NAME

graph – draw a graph

## SYNOPSIS

**graph** [ **–a** *spacing* [ *start* ] ] [ **–b** ] [ **–c** *string* ] [ **–g** *gridstyle* ] [ **–l** *label* ]
[ **–m** *connectmode* ] [ **–s** ] [ **–x** [ **l** ] *lower* [ *upper* [ *spacing* ] ] ]
[ **–y** [ **l** ] *lower* [ *upper* [ *spacing* ] ] ] [ **–h***fraction* ] [ **–w***fraction* ] [ **–r***fraction* ]
[ **–u***fraction* ] [ **–t** ] ...

## DESCRIPTION

**graph** with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the **plot**(1G) filters.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes " **in which case they may be empty or contain blanks and numbers; labels never contain newlines.**

A legend indicating grid range is produced with a grid unless the –s option is present.

## OPTIONS

Each option is recognized as a separate argument.

**–a** *spacing* [ *start* ]
Supply abscissas automatically (they are missing from the input); *spacing* is the spacing (default 1). *start* is the starting point for automatic abscissas (default 0 or lower limit given by **–x**).

**–b** Break (disconnect) the graph after each label in the input.

**–c** *string*
*String* is the default label for each point.

**–g** *gridstyle*
*Gridstyle* is the grid style: 0 no grid, 1 frame with ticks, 2 full grid (default).

**–l** *label* is label for graph.

**–m** *connectmode*
Mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

**–s** Save screen, don't erase before plotting.

**–x** [ **l** ] *lower* [ *upper* [ *spacing*
If **l** is present, *x* axis is logarithmic. *lower* and *upper* are lower (and upper) *x* limits. *spacing*, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

**–y** [ **l** ] *lower* [ *upper* [ *spacing*
If **l** is present, *y* axis is logarithmic. *lower* and *upper* are lower (and upper) *y* limits. *spacing*, if present, is grid spacing on *y* axis. Normally these quantities are determined automatically.

**–h***fraction*
*fraction* of space for height.

**–w***fraction*
*fraction* of space for width.

**–r***fraction*
*fraction* of space to move right before plotting.

**–u***fraction*
*fraction* of space to move up before plotting.

**–t** Transpose horizontal and vertical axes. (Option **–x** now applies to the vertical axis.)

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

**plot**(1G), **spline**(1G)

BUGS

**graph** stores all points internally and drops those for which there isn't room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

NAME

grep, egrep, fgrep – search a file for a string or regular expression

SYNOPSIS

**grep** [ **–bchilnsvw** ] [ **–e** *expression* ] [ *filename*... ]

**egrep** [ **–bchilnsvw** ] [ **–e** *expression* ] [ **–f** *filename* ] [ *expression* ] [ *filename*... ]

**fgrep** [ **–bchilnsvwx** ] [ **–e** *string* ] [ **–f** *filename* ] [ *string* ] [ *filename*... ]

SYSTEM V SYNOPSIS

**/usr/5bin/grep** [ **–bchilnsvw** ] [ **–e** *expression* ] [ *filename*... ]

DESCRIPTION

Commands of the **grep** family search the input *filename*s (the standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. **grep** patterns are limited regular expressions in the style of **ed**(1). **egrep** patterns are full regular expressions including alternation. **fgrep** patterns are fixed strings — no regular expression metacharacters are supported.

In general, **egrep** is the fastest of these programs.

Take care when using the characters '$', '*', [, '^', '|', '(', ')', and '\' in the *expression*, as these characters are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

When any of the **grep** utilities is applied to more than one input file, the name of the file is displayed preceding each line which matches the pattern. The filename is not displayed when processing a single file, so if you actually want the filename to appear, use **/dev/null** as a second file in the list.

OPTIONS

**–b** Precede each line by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

**–c** Display a count of matching lines rather than displaying the lines which match.

**–h** Do not display filenames.

**–i** Ignore the case of letters in making comparisons — that is, upper and lower case are considered identical.

**–l** List only the names of files with matching lines (once) separated by NEWLINE characters.

**–n** Precede each line by its relative line number in the file.

**–s** Work silently, that is, display nothing except error messages. This is useful for checking the error status.

**–v** Invert the search to only display lines that *do not* match.

**–w** Search for the expression as a word as if surrounded by \< and \>. This applies to **grep** only.

**–x** Display only those lines which match exactly — that is, only lines which match in their entirety This applies to **fgrep** only.

**–e** *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a '–'.

**–e** *string*
For **egrep** the argument is a literal character *string*.

**–f** *filename*
Take the regular expression (**egrep**) or a list of strings separated by NEWLINE (**fgrep**) from *filename*.

**SYSTEM V OPTIONS**

The **–s** option to **grep** indicates that error messages for nonexistent or unreadable files should be suppressed, not that all messages *except* for error messages should be suppressed.

**REGULAR EXPRESSIONS**

The following *one-character* regular expressions match a *single* character:

*c* An ordinary character (*not* one of the special characters discussed below) is a one-character regular expression that matches that character.

\c A backslash (\) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

- '.', '*', '[', and '\' (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([ ]).
- '^' (caret or circumflex), which is special at the *beginning* of an *entire* regular expression, or when it immediately follows the left of a pair of square brackets ([ ]).
- **$** (currency symbol), which is special at the *end* of an entire regular expression.

A backslash followed by one of '<', '>', '(', ')', '{', or '}', represents a special operator in the regular expression; see below.

. A '.' (period) is a one-character regular expression that matches any character except NEWLINE.

[ *string* ]

A non-empty string of characters enclosed in square brackets is a one-character regular expression that matches *any one* character in that string. If, however, the first character of the string is a '^' (a circumflex or caret), the one-character regular expression matches any character *except* NEWLINE and the remaining characters in the string. The '^' has this special meaning *only* if it occurs first in the string. The '–' (minus) may be used to indicate a range of consecutive ASCII characters; for example, **[0–9]** is equivalent to **[0123456789]**. The '–' loses this special meaning if it occurs first (after an initial '^', if any) or last in the string. The ']' (right square bracket) does not terminate such a string when it is the first character within it (after an initial '^', if any); that is, **[ ]a–f]** matches either ']' (a right square bracket ) or one of the letters **a** through **f** inclusive. The four characters '.', '*', '[ ', and '\' stand for themselves within such a string of characters.

The following rules may be used to construct regular expressions:

* A one-character regular expression followed by '*' (an asterisk) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

\(and\) A regular expression enclosed between the character sequences \( and \) matches whatever the unadorned regular expression matches. This applies only to **grep**.

*n* The expression *n* matches the same string of characters as was matched by an expression enclosed between \( and \) *earlier* in the same regular expression. Here *n* is a digit; the subexpression specified is that beginning with the *n*th occurrence of \( counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

**Concatenation**

The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

\< The sequence \< in a regular expression constrains the one-character regular expression immediately following it only to match something at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character not one of these.

\> The sequence \> in a regular expression constrains the one-character regular expression immediately following it only to match something at the end of a "word"; that is, either at the end of a line, or just before a character which is neither a letter, digit, nor underline.

\{*m*\}
\{*m*,\}
\{*m*,*n*\} A regular expression followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} matches a **range** of occurrences of the regular expression. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

^ A circumflex or caret (^) at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.

$ A currency symbol ($) at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line.

The construction

**example%** ^*entire regular expression* **$**

constrains the entire regular expression to match the entire line.

**egrep** accepts regular expressions of the same sort **grep** does, except for \(, \), *n*, \<, \>, \{, and \}, with the addition of:

* A regular expression (not just a one-character regular expression) followed by '*' (an asterisk) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

\+ A regular expression followed by '+' (a plus sign) is a regular expression that matches *one* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

? A regular expression followed by '?' (a question mark) is a regular expression that matches *zero* or *one* occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

| Alternation: two regular expressions separated by '|' or NEWLINE match either a match for the first or a match for the second.

() A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is '[ ]' (character classes), then '*' '+' '?' **(closures),**then concatenation, then '|' **(alternation)**and NEWLINE.

EXAMPLES

Search a file for a fixed string using **fgrep**:

**example% fgrep intro /usr/share/man/man3/*.3***

Look for character classes using **grep**:

**example% grep '[1-8]([CJMSNX])' /usr/share/man/man1/*.1**

Look for alternative patterns using **egrep**:

**example% egrep '(Sally|Fred) (Smith|Jones|Parker)' telephone.list**

To get the filename displayed when only processing a single file, use **/dev/null** as the second file in the list:

```
example% grep 'Sally Parker' telephone.list /dev/null
```

FILES

**/dev/null**

SEE ALSO

**awk**(1), **ed**(1), **ex**(1), **sh**(1), **vi**(1), **sed**(1V)

BUGS

Lines are limited to 1024 characters by **grep**; longer lines are truncated.

The combination of **–l** and **–v** options does *not* produce a list of files in which a regular expression is not found. To get such a list, use the Bourne shell construct:

```
for filename in *
do
        if [ `grep "re" $filename | wc -l` -eq 0 ]
        then
                echo $filename
        fi
done
```

or the C shell construct:

```
foreach filename (*)
        if (`grep "re" $filename | wc -l` == 0) echo $filename
end
```

Ideally there should be only one **grep**.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

NAME

groups – display a user's group memberships

SYNOPSIS

**groups [ user ... ]**

DESCRIPTION

With no arguments, **groups** displays the groups to which you belong; else it displays the groups to which the **user** belongs. Each user belongs to a group specified in the password file **/etc/passwd** and possibly to other groups as specified in the file **/etc/group**. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

When a new file is created it is given the group of the containing directory.

FILES

**/etc/passwd**
**/etc/group**

SEE ALSO

**getgroups**(2)

## NAME

head – display first few lines of specified files

## SYNOPSIS

**head** [ *–n* ] [ *filename*... ]

## DESCRIPTION

**head** copies the first *n* lines of each *filename* to the standard output. If no *filename* is given, **head** copies lines from the standard input. The default value of *n* is 10 lines.

When more than one file is specified, the start of each file which looks like:

==>*filename*<==

Thus, a common way to display a set of short files, identifying each one, is:

**example% head** –9999 *filename1 filename2* ...

## EXAMPLE

The following example:

**example% head -4 /usr/share/man/man1/{cat,head,tail}.1***

produces:

```
==> /usr/share/man/man1/cat.1v <==
.TH CAT 1V "2 June 1983"
.SH NAME
cat – concatenate and display
.SH SYNOPSIS

==> /usr/share/man/man1/head.1 <==
.TH HEAD 1 "24 August 1983"
.SH NAME
head – display first few lines of specified files
.SH SYNOPSIS

==> /usr/share/man/man1/tail.1 <==
.TH TAIL 1  "27 April 1983"
.SH NAME
tail – display the last part of a file
.SH SYNOPSIS
```

## SEE ALSO

**cat**(1V), **more**(1), **tail**(1)

## NAME

help – ask for help regarding SCCS errors or warnings

## SYNOPSIS

**/usr/sccs/help** [ *arguments* ]

## DESCRIPTION

**help** finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, **help** will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **ge6**, for message 6 from the **get** command).

type 2 Does not contain numerics (as a command, such as **get**(1)).

type 3 Is all numeric (for example, **212**).

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try '**/usr/sccs/help stuck**'.

## FILES

**/usr/lib/help** directory containing files of message text.
**/usr/sccs/help**

## SEE ALSO

**get**(1)

NAME

help_viewer – SunView application providing help with applications and desktop

SYNOPSIS

**/usr/lib/help_viewer** [ *options* ]

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**help_viewer** allows you to quickly access documentation about SunView applications and the SunView Desktop. This help consists of intermixed text and graphics displayed in a window called the Help Viewer.

There are two ways **help_viewer** can be invoked. One is as a stand-alone SunView application, where **help_viewer** starts up with a list of the applications that it contains documentation for. The second way to reach **help_viewer** is by clicking on the More Help button in a Spot Help window. In this case, the Help Viewer comes up with text specific to the current application and context.

The documentation within **help_viewer** is extendable, but as shipped it includes handbooks for the DeskTop, **mailtool**(1), **shelltool**(1), **textedit**(1), organizer, SunPC, and itself (**help_viewer**).

The available documentation depends only on the existence of the appropriate files in the directories specified under FILES.

The user moves between the various pages of help with the assistance of hypertext *links*. Links are symbolic connections between pages of text. The current convention at Sun is to use underlined text to indicate the presence of a link. When the user double-clicks on a link, the text associated with the topic indicated by the link is shown in the Help Viewer. There are links in many places to make it quick and easy to go from place to place within the **help_viewer** database.

At the lower levels in the hierarchy of help files, many of the topics contain more than one page of text, and in these cases a link to the next page and to the previous page is available at the upper-right corner of the Help Viewer which allows you to page through the document.

The current position within the hierarchy of text is indicated by the links at the upper-left corner of the Help Viewer. The last link in the list is the level just above your current position.

OPTIONS

The standard SunView options for window size, position, fonts, and other options are allowed. See **sunview**(1) for details.

**–dir** *dirname*

Name of help directory

**–file** *filename* [#]

Name of startup file relative to help directory (or **/usr/lib/help** by default). # is a page number separated from the filename by a SPACE. If # is omitted, the first page is shown.

FILES

**/usr/lib/help** directory containing miscellaneous help files

The files in **/usr/lib/help** are used by the **help** and the **help_viewer** facilities, and the SCCS **help**(1) facility. Directories within **/usr/lib/help** named after SunView applications and the DeskTop contain specific information used by **help_viewer**. See **help_viewer**(5) for information about the files in these directories.

SEE ALSO

**help**(1), **mailtool**(1), **shelltool**(1), **textedit**(1), **help_viewer**(5)

DIAGNOSTICS

**help_viewer**(1) displays a pop-up error window if it cannot find the file required to show the requested help.

**NAME**

hostid – print the numeric identifier of the current host

**SYNOPSIS**

**hostid**

**DESCRIPTION**

The **hostid** command prints the identifier of the current host in hexadecimal. This numeric value is unique across all Sun hosts.

**SEE ALSO**

**gethostid**(2)

NAME

hostname – set or print name of current host system

SYNOPSIS

**hostname** [ *name-of-host* ]

DESCRIPTION

The **hostname** command prints the name of the current host, as given before the "login" prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script **/etc/rc.local**.

FILES

**/etc/rc.local**

SEE ALSO

**gethostname**(2)

NAME

iconedit – create and edit images for SunView icons, cursors and panel items

SYNOPSIS

**iconedit** [ *filename* ]

OPTIONS

**iconedit** accepts the standard SunView command-line arguments; see **sunview(1)** for a list.

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**iconedit** is a standard tool provided with the SunView environment. With it you can create and edit small images for use in icons, cursors, panel items, etc. **iconedit** has several subwindows:

- A large drawing area or *canvas* (on the left).
- A small proof area for previewing a life-size version of the image being edited (at the lower right).
- A control panel showing the options available and their current state (at the center right).
- An area for status messages (at the upper right).
- An area containing instructions for the use of the mouse (above the drawing canvas).

Inside the canvas, use the left button to draw and the middle button to erase. As you draw, an enlarged version of the image appears in the canvas, while a life-sized version of the image appears in the proof area. Use the right button to undo the previous operation.

While editing a cursor image, you can try the cursor out against different backgrounds and with different raster operations by moving the cursor into the proof area.

CONTROL PANEL

The large control panel to the right of the canvas contains many items through which you can control **iconedit**. Some items are buttons which allow you to initiate commands, some are text fields which you type into, and some are choice items allowing you select from a range of options. Use the left button to select items. Most items also have a menu which you can invoke with the right button.

There are three text fields: the two at the top labeled **Dir:** and **File:**, and one to the right of the **abc** labeled **Fill:**. A triangular caret points to the current type-in position. Typing RETURN advances the caret to the next text field; you can also move the caret to a text field by selecting the field with the left button.

Each item in the control panel is described below:

**Dir** The current directory.

**File** The current filename. The default is the filename given on the command line. You can request filename completion by pressing ESC. **iconedit** searches the current directory for files whose names begin with the string you entered. If the filename search locates only one file, that file will be loaded in. In addition, typing CTRL-L, CTRL-S, CTRL-B or CTRL-Q are equivalent to pressing the **Load**, **Store**, **Browse**, or **Quit** buttons, respectively.

**Load** **(Button)** Load the canvas from the file named in the **File** field.

**Store** **(Button)** Store the current image in the file named in the **File** field.

**Browse** **(Button)** Display all the images in the current directory in a popup panel. When you select an image with the left button, it will be loaded into the canvas for editing and the browsing panel will be hidden. Pressing browse again will cause the panel to popup again (it will come up immediately if the directory and file fields have not been modified).

**Quit** **(Button)** Terminate processing. Quitting requires confirmation.

**Size** Alter the canvas size. Choices are icon size (64 x 64 pixels) or cursor size (16 x 16 pixels).

**Grid** Display a grid over the drawing canvas, or turn the grid off.

**Clear** **(Button)** Clear the canvas.

**Fill** **(Button)** Fill canvas with current rectangular fill pattern.

**Invert** **(Button)** Invert each pixel represented on the canvas.

**Paintbrush**

Select from among five painting modes. Instructions for each painting mode appear above the canvas. The painting modes are:

**dot** Paint a single dot at a time.

**line** Draw a line. To draw a line on the canvas, point to the first endpoint of the line, and press and hold the left mouse button. While holding the button down, drag the cursor to the second endpoint of the line. Release the mouse button.

**rectangle**

Draw a rectangle. To draw a rectangle on the canvas, point to the first corner of the rectangle and press and hold the left mouse button. While holding the button down, drag the cursor to the diagonally opposite corner of the rectangle. Release the mouse button.

In the control panel, the **Fill** field to the right of the rectangle indicates the current rectangle fill pattern. Any rectangles you paint on the canvas will be filled with this pattern.

**circle** Draw a circle. To draw a circle on the canvas, point to the center of the circle, and press and hold the left mouse button. While holding the button down, drag the cursor to the desired edge of the circle. Release the mouse button.

In the control panel, the **Fill** field to the right of the circle indicates the current circle fill pattern. Any circles you paint on the canvas will be filled with this pattern.

**abc** Insert text. To insert text, move the painting hand to **abc** and type the desired text. Then move the cursor to the canvas and press and hold the left mouse button. A box will appear where the text is to go. Position the box as desired and release the mouse button.

In addition, you can choose the font in which to draw the text. Point at the Fill field to the right of the **abc** and either click the left mouse button to cycle through the available fonts or press and hold the right mouse button to bring up a menu of fonts.

**Load** This is the rasterop to be used when loading a file in from disk. (See the *Pixrect Reference Manual* for details on rasterops).

**Fill** This is the rasterop to be used when filling the canvas. The source for this operation is the rectangle fill pattern, and the destination is the canvas.

**Proof** This is the rasterop to be used when rendering the proof image. The source for this operation is the proof image, and the destination is the proof background.

Proof background

The proof background can be changed to allow you to preview how the image will appear against a variety of patterns. The squares just above the proof area show the patterns available for use as the proof background pattern. To change the proof background, point at the desired pattern and click the left mouse button.

SEE ALSO

**sunview**(1)

*Pixrect Reference Manual*

NAME

id – print the user name and ID, and group name and ID

SYNOPSIS

**id**

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command.

**id** displays your user and group ID, and your username. If your real and effective IDs do not match, both are printed.

SEE ALSO

**getuid**(2)

## NAME

indent – indent and format a C program source file

## SYNOPSIS

**indent** [ *input-file* [ *output-file* ] ] [ [ **–bap** | **–nbap** ] [ **–bacc** | **–nbacc** ] [ **–bad** | **–nbad** ] [ **–bbb** | **–nbbb** ] [ **–bc** | **–nbc** ] [ **–bl** ] [ **–br** ] [ **–bs** | **–nbs** ] [ **–c***n* ] [ **–cd***n* ] [ **–cdb** | **–ncdb** ] [ **–ce** | **–nce** ] [ **–ci***n* ] [ **–cli***n* ] [ **–d***n* ] [ **–di***n* ] [ **–eei** | **–neei** ] [ **–fc1** | **–nfc1** ] [ **–i***n* ] [ **–ip** | **–nip** ] [ **–l***n* ] [ **–lc***n* ] [ **–lp** | **–nlp** ] [ **–pcs** | **–npcs** ] [ **–npro** ] [ **–psl** | **–npsl** ] [ **–sc** | **–nsc** ] [ **–sob** | **–nsob** ] [ **–st** ] [ **–troff** ] [ **–v** | **–nv** ]

## DESCRIPTION

**indent** is a C program formatter. It reformats the C program in the *input-file* according to the switches. The switches which can be specified are described below. They may appear before or after the file names.

Note: if you only specify an *input-file*, the formatting is done "in-place", that is, the formatted file is written back into *input-file* and a backup copy of *input-file* is written in the current directory. If *input-file* is named **/blah/blah/file**, the backup file is named **file.BAK**.

If *output-file* is specified, **indent** checks to make sure it is different from *input-file*.

## OPTIONS

The options listed below control the formatting style imposed by **indent**.

**–bap,–nbap** If **–bap** is specified, a blank line is forced after every procedure body. Default: **–nbap.**

**–bacc,–nbacc** If **–bacc** is specified, a blank line is forced around every conditional compilation block. That is. in front of every **#ifdef** and after every **#endif**. Other blanklines surrounding these will be swallowed. Default: **–nbacc.**

**–bad,–nbad** If **–bad** is specified, a blank line is forced after every block of declarations. Default: **–nbad.**

**–bbb,–nbbb** If **–bbb** is specified, a blank line is forced before every block comment. Default: **–nbbb.**

**–bc,–nbc** If **–bc** is specified, then a NEWLINE is forced after each comma in a declaration. **–nbc** turns off this option. The default is **–bc.**

**–br,–bl** Specifying **–bl** lines up compound statements like this:

```
if (...)
{
                    code
}
```

Specifying **–br** (the default) makes them look like this:

```
if (...) {
                    code
}
```

**–bs,–nbs**

Enable (disable) the forcing of a blank after **sizeof**. Some people believe that **sizeof** should appear as though it were a procedure call (**–nbs**, the default) and some people believe that since **sizeof** is an operator, it should always be treated that way and should always have a blank after it.

**–c***n* The column in which comments on code start. The default is 33.

**–cd***n* The column in which comments on declarations start. The default is for these comments to start in the same column as those on code.

**–cdb,–ncdb**

Enable (disable) the placement of comment delimiters on blank lines. With this option enabled, comments look like this:

```
/*
* this is a comment
*/
```

Rather than like this:

```
/* this is a comment */
```

This only affects block comments, not comments to the right of code. The default is **–cdb .**

**–ce,–nce**

Enables (disables) forcing **else**'s to cuddle up to the immediately preceeding '}'. The default is **–ce.**

**–ci***n* Sets the continuation indent to be *n*. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless **–lp** is in effect. **–ci** defaults to the same value as **–i.**

**–cli***n* Cause case labels to be indented *n* tab stops to the right of the containing **switch** statement. **-cli0.5** causes case labels to be indented half a tab stop. The default is **–cli0 .**

**–d***n* Control the placement of comments which are not to the right of code. The default **–d1** means that such comments are placed one indentation level to the left of code. Specifying **–d0** lines up these comments with the code. See the section on comment indentation below.

**–di***n* Specify the indentation, in character positions, from a declaration keyword to the following identifier. The default is **–di16 .**

**–eei,–neei**

If **–eei** is specified, and extra expression indent is applied on continuation lines of the expression part of **if**( ) and **while**( ). These continuation lines will be indented one extra level — twice instead of just once. This is to avoid the confusion between the continued expression and the statement that follows the **if**( ) or **while**( ). Default: **–neei.**

**–fc1,–nfc1**

Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading '/' is in column 1 have been carefully hand formatted by the programmer. In such cases, **–nfc1** should be used. The default is **–fc1.**

**–i***n* The number of spaces for one indentation level. The default is 4.

**–ip,–nip**

Enables (disables) the indentation of parameter declarations from the left margin. The default is **–ip .**

**–l***n* Maximum length of an output line. The default is 78.

**–lc***n* Sets the line length for block comments to *n*. It defaults to being the same as the usual line length as specified with **–l.**

**–lp,–nlp**

Lines up code surrounded by parenthesis in continuation lines. If a line has a left paren which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left parenthesis. For example, here is how a piece of continued code looks with **–nlp** in effect:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

With **-lp** in effect (the default) the code looks somewhat clearer:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

Inserting a couple more NEWLINE characters we get:

```
p1 = first_procedure(second_procedure(p2,
                                      p3),
                     third_procedure(p4,
                                     p5));
```

**-npro** Ignore the profile files, **./.indent.pro** and **~/.indent.pro**.

**-pcs , -npcs**
If true (**-pcs**) all procedure calls will have a space inserted between the name and the '('. The default is **-npcs**

**-psl , -npsl**
If true (**-psl**) the names of procedures being defined are placed in column 1 — their types, if any, will be left on the previous lines. The default is **-psl**.

**-sc,-nsc**
Enables (disables) the placement of asterisks ('*'s) at the left edge of all comments.

**-sob,-nsob**
If **-sob** is specified, indent will swallow optional blank lines. You can use this to get rid of blank lines after declarations. Default: **-nsob**

**-st** **indent** takes its input from the standard input, and put its output to the standard output.

**-T** *typename*
Add *typename* to the list of type keywords. Names accumulate: **-T** can be specified more than once. You need to specify all the typenames that appear in your program that are defined by **typedef**s — nothing will be harmed if you miss a few, but the program won't be formatted as nicely as it should. This sounds like a painful thing to have to do, but it is really a symptom of a problem in C: **typedef** causes a syntactic change in the language and *indent* cannot find all **typedef**s.

**-troff** Causes **indent** to format the program for processing by **troff**. It will produce a fancy listing in much the same spirit as **vgrind**. If the output file is not specified, the default is standard output, rather than formatting in place. The usual way to get a **troff**ed listing is with the command

**indent -troff** *program.c* **| troff -mindent**

**-v,-nv** -v turns on "verbose" mode, **-nv** turns it off. When in verbose mode, **indent** reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is **-nv**.

## FURTHER DESCRIPTION

You may set up your own "profile" of defaults to **indent** by creating a file called **.indent.pro** in either your login directory or the current directory and including whatever switches you like. An **.indent.pro** in the current directory takes precedence over the one in your login directory. If **indent** is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, always override profile switches. The switches should be separated by SPACE, TAB, or NEWLINE characters.

### Comments

Boxed **indent** assumes that any comment with a dash or star immediately after the start of comment (that is, '/*-'or'/**') is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that its indentation may be adjusted to account for the change in indentation of the first line of the comment.

Straight text — All other comments are treated as straight text. **indent** fits as many words (separated by SPACE, TAB, or NEWLINE characters) on a line as possible. Blank lines break paragraphs.

**Comment indentation**

If a comment is on a line with code it is started in the "comment column", which is set by the **–c***n* command line parameter. Otherwise, the comment is started at *n* indentation levels less than where code is currently being placed, where *n* is specified by the **–d***n* command line parameter. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

**Preprocessor lines**

In general, **indent** leaves preprocessor lines alone. The only reformatting that it will do is to straighten up trailing comments. It leaves imbedded comments alone. Conditional compilation (**#ifdef...#endif**) is recognized and **indent** attempts to correctly compensate for the syntactic peculiarities introduced.

**C syntax**

**indent** understands a substantial amount about the syntax of C, but it has a "forgiving" parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like:

```
#define forever for(;;)
```

is handled properly.

## FILES

| | |
|---|---|
| **./.indent.pro** | profile file |
| **~/.indent.pro** | profile file |
| **/usr/share/lib/tmac/tmac.indent** | troff macro package for '**indent –troff**' output. |

## SEE ALSO

**ls**(1V), **troff**(1)

## BUGS

**indent** has even more switches than **ls**(1V).

A common mistake that often causes grief is typing:

```
indent *.c
```

to the shell in an attempt to indent all the C programs in a directory. This is probably a bug, not a feature.

The **–bs** option splits an excessively fine hair.

## NAME

indxbib – create an inverted index to a bibliographic database

## SYNOPSIS

**indxbib** *database-file...*

## AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**indxbib** makes an inverted index to the named *database-file* (which must reside within the current directory), typically for use by **lookbib**(1) and **refer**(1). A *database* contains bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '**%**', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '**%**'.

**indxbib** is a shell script that calls two programs: **/usr/lib/refer/mkey** and **/usr/lib/refer/inv**. **mkey** truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed. See **refer** in *Formatting Documents* for details.

**indxbib** creates an entry file (with a **.ia** suffix), a posting file (**.ib**), and a tag file (**.ic**), in the working directory.

## FILES

**/usr/lib/refer/mkey**
**/usr/lib/refer/inv**

| | |
|---|---|
| *x*.**ia** | entry file |
| *x*.**ib** | posting file |
| *x*.**ic** | tag file |
| *x*.**ig** | reference file |

## SEE ALSO

**addbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1)

*Formatting Documents*

## BUGS

All dates should probably be indexed, since many disciplines refer to literature written in the 1800s or earlier.

**indxbib** does not recognize pathnames.

NAME

inline – in-line procedure call expander

SYNOPSIS

**/usr/lib/inline** [ **–w** ] [ **–v** ] [ **-o** *outputfile* ] [ **–i** *inlinefile* ] ... [ *cpu-option* ] [ *fpu-option* ] *filename* ...

DESCRIPTION

**inline** expands assembly language calls in the indicated source files into copies of the corresponding procedure bodies obtained from an *inlinefile* specified with the **–i** option. If no *inlinefile* is specified, the source files are simply concatenated and written to the standard output. If no source files are specified, the input is read from the standard input.

Inline itself is little more than a sed script. Almost all of the benefit produced is derived from subsequent peephole optimization.

OPTIONS

**–w** Display warnings for duplicate definitions on the standard error.

**–v** Verbose. Display the names of routines that were actually in-line expanded in the sourcefile on the standard error.

**–o** *outputfile*
write output to the indicated file; standard output by default.

**–i** *inlinefile*
Read in-line code templates from *inlinefile*.

*cpu-option*
Specify templates for the machine architecture of a Sun-2 or Sun-3 system. If this option is omitted, the proper template for the host architecture is used. Can be one of:

**–mc68010** expand **.mc68010** code templates
**–mc68020** expand **.mc68020** code templates

*fpu-option*
Specify a floating-point processor option for a Sun-2 or Sun-3 system. Can be one of:

**–fsoft** expand **.fsoft** code templates (the default)
**–fswitch** expand **.fswitch** code templates
**–fsky** expand **.fsky** code templates (**–mc68010** only)
**–f68881** expand **.f68881** code templates (**–mc68020** only)
**–ffpa** expand **.ffpa** code templates (**–mc68020** only)

USAGE

Each *inlinefile* contains one or more labeled assembly language templates of the form:

*inline-directive*
...
*instructions*
...
*.end*

where the *instructions* constitute an in-line expansion of the named routine. An *inline-directive* is a command of the form:

**.inline** *identifier*, *argsize*

This declares a block of code for the routine named by *identifier*, with *argsize* bytes of arguments. (*argsize* is optional on Sun-4 systems). Calls to the named routine are replaced by the code in the in-line template.

For Sun-2 and Sun-3 systems, the following additional forms are recognized:

| | |
|---|---|
| **.mc68010** | *identifier*, *argsize* |
| **.mc68020** | *identifier*, *argsize* |
| **.fsoft** | *identifier*, *argsize* |
| **.fswitch** | *identifier*, *argsize* |
| **.fsky** | *identifier*, *argsize* |
| **.f68881** | *identifier*, *argsize* |
| **.ffpa** | *identifier*, *argsize* |

These forms are similar to **.inline**, with the addition of a CPU or FPU specification. The template is only expanded if the specified target system matches the value of the target CPU or FPU type, as determined by the command-line options, or if none were given, by the type of the host system.

Multiple templates are permitted; matching templates after the first are ignored. Duplicate templates may be placed in order of decreasing performance of the corresponding hardware; thus the most efficient usable version will be selected.

**Coding Conventions for all Sun Systems**

In-line templates should be coded as expansions of C-compatible procedure calls, with the difference that the return address cannot be depended upon to be in the expected place, since no call instruction will have been executed. See FILES, below, for examples.

In-line templates must conform to standard Sun parameter passing and register usage conventions, as detailed below. They must not call routines that violate these conventions; for example, assembly language routines such as **setjmp**(3) may cause problems.

Registers other than the ones mentioned below must not be used or set.

Branch instructions in an in-line template may only transfer to numeric labels (**1f**, **2b**, and so on) defined within the in-line template. No other control transfers are allowed.

Only opcodes and addressing modes generated by Sun compilers are guaranteed to work. Binary encodings of instructions are not supported.

**Coding Conventions for Sun-2 and Sun-3**

Arguments are passed in 32-bit aligned memory locations starting at **sp@**. Note that there is no return address on the stack, since no **jbsr** instruction will have been executed.

Results are returned in **d0** or **d0/d1**.

The following registers may be used as temporaries: registers **a0**, **a1**, **d0**, and **d1** on the MC68010 and MC68020; registers **fp0** and **fp1** on the MC68881; registers **fpa0** through **fpa3** on the Sun Floating-Point Accelerator. No other registers may be used.

The template must delete exactly *argsize* bytes from the stack. This is to enable **inline** to deal with autoincrement and autodecrement addressing modes, which in turn are used by **c2** to delimit the lifetimes of stack temporaries.

The stack must not underflow the level of the last argument.

Use j*cc* branch mnemonics instead of b*cc*. The b*cc* ops are span limited and will fail if retargeted to a label whose span overflows the branch displacement field.

**Coding Conventions for Sun-4 Systems**

Arguments are passed in registers **%o0**-**%o5**, followed by memory locations starting at **[%sp+0x5c]**. **%sp** is guaranteed to be 64-bit aligned. The contents of **%o7** are undefined, since no call instruction will have been executed.

Results are returned in **%o0** or **%f0**/**%f1**.

Registers **%o0**-**%o5** and **%f0**-**%f31** may be used as temporaries.

Integral and single-precision floating-point arguments are 32-bit aligned.

Double-precision floating-point arguments are guaranteed to be 64-bit aligned if their offsets are multiples of 8.

Each control-transfer instruction (branches and calls) must be immediately followed by a nop.

Call instructions must include an extra (final) argument which indicates the number of registers used to pass parameters to the called routine.

Note that for Sun-4 systems, the instruction following an expanded 'call' is inserted by **inline** *before* the expanded code to preserve the semantics of the call's delay slot.

FILES

| | |
|---|---|
| **/usr/lib/inline** | in-line procedure call expander |
| **/usr/lib/fsoft/libm.il** | in-line templates for software floating point (Sun-2 and Sun-3 only) |
| **/usr/lib/fswitch/libm.il** | in-line templates for switched floating point (Sun-2 and Sun-3 only) |
| **/usr/lib/fsky/libm.il** | in-line templates for Sky FFP (Sun-2 only) |
| **/usr/lib/f68881/libm.il** | in-line templates for Motorola 68881 (Sun-3 only) |
| **/usr/lib/ffpa/libm.il** | in-line templates for Sun FPA (Sun-3 only) |

WARNING

**inline** does not check for violations of the coding conventions described above.

NAME

input_from_defaults, defaults_from_input – update the current state of the mouse and keyboard from the defaults database, and vice versa

SYNOPSIS

**input_from_defaults**
**defaults_from_input**

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**input_from_defaults** updates various parameters controlling mouse- and keyboard-processing on the machine on which it is run. It should be used on systems which are running the SunView window system. The parameters control the distribution of function keys on the keyboard, the assignment of buttons on the mouse, the scaling of mouse-to-cursor motion, and the effect of two filters on mouse-motion originally provided to compensate for defective mice. The new values are taken from the defaults database, starting with the file **.defaults** in the user's home directory.

**defaults_from_input** is the inverse operation to **input_from_defaults**. It updates a the user's private defaults database (used by **defaultsedit**(1)) to reflect the current state of kernel input parameters listed above.

FILES

**$HOME/.defaults**
**/usr/lib/defaults/*.d**

SEE ALSO

**defaultsedit**(1)

*SunView 1 Beginner's Guide*

BUGS

**input_from_defaults** should be targetable to any user's **.defaults** file.

NAME

install – install files

SYNOPSIS

**install** [ **–cs** ] [ **–g** *group* ] [ **–m** *mode* ] [ **–o** *owner* ] *file1 file2*

**install** [ **–cs** ] [ **–g** *group* ] [ **–m** *mode* ] [ **–o** *owner* ] *file ... directory*

**install –d** [ **–g** *group* ] [ **–m** *mode* ] [ **–o** *owner* ] *directory*

AVAILABILITY

This command is available with the *Install Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

Install is used within makefiles to copy new versions of files into a destination directory and to create the destination directory itself.

The first two forms are similar to the **cp**(1) command with the addition that executable files can be stripped during the copy and the owner, group, and mode of the installed file(s) can be given.

The third form can be used to create a destination directory with the required owner, group and permissions.

Note: **install** uses no special privileges to copy files from one place to another. The implications of this are:

- You must have permission to read the files to be installed.
- You must have permission to copy into the destination file or directory.
- You must have permission to change the modes on the final copy of the file if you want to use the **–m** option to change modes.
- You must be superuser if you want to use the **–o** option to change ownership.

OPTIONS

**–g** *group*

Set the group ownership of the installed file or directory. (**staff** by default)

**–m** *mode*

Set the mode for the installed file or directory. (**0755** by default)

**–o** *owner*

Set the owner of the installed file or directory. (**root** by default)

**–c** Copy files. In fact **install** *always* copies files, but the **–c** option is retained for backwards compatibility with old shell scripts that might otherwise break.

**–s** Strip executable files as they are copied.

**–d** Create a directory. Missing parent directories are created as required as in **mkdir –p**. If the directory already exists, the owner, group and mode will be set to the values given on the command line.

SEE ALSO

**chmod**(1V), **chgrp**(1), **cp**(1), **mkdir**(1), **strip**(1), **chown**(8)

NAME

ipcrm – remove a message queue, semaphore set, or shared memory ID

SYNOPSIS

**ipcrm** [ *primitives* ]

DESCRIPTION

**ipcrm** removes one or several messages, semaphores, or shared memory identifiers, as specified by the following *primitives*:

**–q** *msqid*
removes the message queue identifier *msqid* from the system and destroys the message queue and data structures associated with it.

**–m** *shmid*
removes the shared memory identifier *shmid* from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.

**–s** *semid*
removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structures associated with it.

**–Q** *msgkey*
removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structures associated with it.

**–M** *shmkey*
removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structures associated with it are destroyed after the last detach.

**–S** *semkey*
removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structures associated with it.

The identifiers and keys may be found by using **ipcs**(1).

The details of removing identifiers are described in **msgctl**(2), **shmctl**(2), and **semctl**(2) in the sections detailing the **IPC_RMID** command.

SEE ALSO

**ipcs**(1), **msgctl**(2), **msgget**(2), **semctl**(2), **semget**(2), **semop**(2), **shmctl**(2), **shmget**(2), **shmop**(2)

**NAME**

ipcs – report interprocess communication facilities status

**SYNOPSIS**

**ipcs** [ *primitives* ]

**DESCRIPTION**

**ipcs** prints information about active interprocess communication facilities as specified by the *primitives* shown below. If no *primitives* are given, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system.

**Command – Line Primitives**

If any of the *primitives* **–q**, **–m**, or **–s** are specified, information about only indicated facilities is printed. If none of these are specified, information about all three is printed.

**–q** Print information about active message queues.

**–m** Print information about active shared memory segments.

**–s** Print information about active semaphores.

**–b** Print the biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for the meaning of columns in a listing.

**–c** Print creator's login name and group name. See below.

**–o** Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

**–p** Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments)See below.

**–t** Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last **msgsnd** and last **msgrcv** (see **msgop**(2)) on message queues, last **shmat** and last **shmdt** (see **shmop**(2)) on shared memory, last **semop**(2) on semaphores.) See below.

**–a** Use all display *primitive*s. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

**–C** *corefile*
Use the file *corefile* in place of **/dev/kmem**.

**–N** *namelist*
The argument will be taken as the name of an alternate *filenamelist* (**/vmunix** is the default).

The column headings and the meaning of the columns in an **ipcs** listing are given below; the letters in parentheses indicate the *primitives* that cause the corresponding heading to appear; **all** means that the heading always appears. Note: these *primitives* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

**T** (all) Type of the facility:
- **q** message queue
- **m** shared memory segment
- **s** semaphore

**ID** (all) The identifier for the facility entry.

**KEY** (all) The key used as an argument to **msgget**(2), **semget**(2), or **shmget**(2) to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

**MODE** (all) The facility access modes and flags: The mode consists of 11 characters that are

interpreted as follows:

The first two characters are:

**R** If a process is waiting on a **msgrcv**.
**S** If a process is waiting on a **msgsnd**.
**D** If the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it.
**C** If the associated shared memory segment is to be cleared when the first attach is executed.
**–** If the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

**r** If read permission is granted.
**w** If write permission is granted.
**a** If alter permission is granted.
**—** If the indicated permission is *not* granted.

| | | |
|---|---|---|
| **OWNER** | (all) | The login name of the owner of the facility entry. |
| **GROUP** | (all) | The group name of the group of the owner of the facility entry. |
| **CREATOR** | (a,c) | The login name of the creator of the facility entry. |
| **CGROUP** | (a,c) | The group name of the group of the creator of the facility entry. |
| **CBYTES** | (a,o) | The number of bytes in messages currently outstanding on the associated message queue. |
| **QNUM** | (a,o) | The number of messages currently outstanding on the associated message queue. |
| **QBYTES** | (a,b) | The maximum number of bytes allowed in messages outstanding on the associated message queue. |
| **LSPID** | (a,p) | The process ID of the last process to send a message to the associated queue. |
| **LRPID** | (a,p) | The process ID of the last process to receive a message from the associated queue. |
| **STIME** | (a,t) | The time the last message was sent to the associated queue. |
| **RTIME** | (a,t) | The time the last message was received from the associated queue. |
| **CTIME** | (a,t) | The time when the associated entry was created or changed. |
| **NATTCH** | (a,o) | The number of processes attached to the associated shared memory segment. |
| **SEGSZ** | (a,b) | The size of the associated shared memory segment. |
| **CPID** | (a,p) | The process ID of the creator of the shared memory entry. |
| **LPID** | (a,p) | The process ID of the last process to attach or detach the shared memory segment. |
| **ATIME** | (a,t) | The time the last attach was completed to the associated shared memory segment. |
| **DTIME** | (a,t) | The time the last detach was completed on the associated shared memory segment. |
| **NSEMS** | (a,b) | The number of semaphores in the set associated with the semaphore entry. |
| **OTIME** | (a,t) | The time the last semaphore operation was completed on the set associated with the semaphore entry. |

FILES

| | |
|---|---|
| **/vmunix** | system namelist |
| **/dev/kmem** | memory |
| **/etc/passwd** | user names |
| **/etc/group** | group names |

SEE ALSO

**ipcrm**(1), **msgop**(2), **semctl**(2), **semget**(2), **semop**(2), **shmctl**(2), **shmget**(2), **shmop**(2)

**BUGS**

Things can change while **ipcs** is running; the picture it gives is only a close approximation to reality.

NAME

join – relational database operator

SYNOPSIS

**join** [ **–a***n* ] [ **–e** *string* ] [ **–j** [**1**|**2**] *m* ] [ **–o** *list* ] [ **–t***c* ] *filename1 filename2*

DESCRIPTION

**join** forms, on the standard output, a join of the two relations specified by the lines of *filename1* and *filename2*. If *filename1* is '–', the standard input is used.

*filename1* and *filename2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined — normally the first in each line.

There is one line in the output for each pair of lines in *filename1* and *filename2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *filename1*, then the rest of the line from *filename2*.

The default input field separators are SPACE, TAB, and NEWLINE characters. If the default input field separators are used, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

OPTIONS

**–a***n* The parameter *n* can be one of the values:

1 Produce a line for each unpairable line in *filename1*.

2 Produce a line for each unpairable line in *filename2*.

3 Produce a line for each unpairable line in both *filename1* and *filename2*.

The normal output is also produced.

**–e** *string*

Replace empty output fields by *string*.

**–j**[**1** | **2**]*m*

The **j** may be immediately followed by *n*, which is either a **1** or a **2**. If *n* is missing, the join is on the *m*'th field of both files. If *n* is present, the join is on the *m*'th field of file *n*, and the first field of the other. Note: **join** counts fields from 1 instead of 0 as **sort**(1V) does.

**–o** *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested. Note: **join** counts fields from 1 instead of 0 like **sort** does.

**–t***c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

**join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group**

SEE ALSO

**awk**(1), **comm**(1), **look**(1), **sort**(1V), **uniq**(1)

BUGS

With default field separation, the collating sequence is that of **sort –b**; with **–t**, the sequence is that of a plain sort.

The conventions of **join**, **sort**, **comm**, **uniq**, **look**, and **awk** are wildly incongruous.

Filenames that are numeric may cause conflict when the **–o** option is used right before listing filenames.

**NAME**

keylogin – decrypt and store secret key

**SYNOPSIS**

**keylogin**

**DESCRIPTION**

**keylogin** prompts the user for their login password, and uses it do decrypt the user's secret key stored in the **publickey**(5) database. Once decrypted, the user's key is stored by the local key server process **keyserv**(8C) to be used by any secure network services, such as NFS.

Normally, **login**(1) does this work when the user logs onto the system, but running **keylogin** may be necessary if the user did not type a password to **login**(1).

**SEE ALSO**

**chkey**(1), **login**(1), **publickey**(5), **keyserv**(8C), **newkey**(8)

NAME

kill – send a signal to a process, or terminate a process

SYNOPSIS

**kill** [ *–signal* ] *pid* ...
**kill –l**

DESCRIPTION

**kill** sends the **TERM** (terminate, 15) signal to the processes with the specified *pid*s. If a signal name or number preceded by '–' is given as first argument, that signal is sent instead of terminate. The signal names are listed by using the –l option, and are as given in **/usr/include/signal.h**, stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal, so '**kill –9** ...' is a sure **kill**, as the **KILL** (9) signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (that is, processes resulting from the current login) are signaled (but beware: this works only if you use **sh**(1); not if you use **csh**(1).) Negative process numbers also have special meanings; see **kill**(2V) for details. The killed processes must belong to the current user unless he is the super-user.

To shut the system down and bring it up single user the super-user may send the initialization process a **TERM** (terminate) signal by '**kill 1**'; see **init**(8). To force **init** to close and open terminals according to what is currently in **/etc/ttytab** use '**kill**–HUP 1' (sending a hangup, signal 1).

The shell reports the process number of an asynchronous process started with '**&**' (run in the background). Process numbers can also be found by using **ps**(1).

**kill** is built in to **csh**(1); it allows job specifiers, such as '**kill % ...**', in place of **kill** arguments. See **csh**(1) for details.

OPTIONS

**–l** Display a list of signal names.

FILES

**/etc/ttytab**
**/usr/include/signal.h**

SEE ALSO

**csh**(1), **ps**(1), **kill**(2V), **sigvec(2)**, **init**(8)

BUGS

A replacement for '**kill 0**' for **csh**(1) users should be provided.

## NAME

last – indicate last logins by user or terminal

## SYNOPSIS

**last** [ *–number* ][ **–f** *filename* ] [ *name...* ] [ *tty...* ]

## DESCRIPTION

**last** looks back in the **wtmp** file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example **last 0** is the same as **last***tty0*. If multiple arguments are given, the information which applies to any of the arguments is printed. For example **last root console** would list all of "root's" sessions as well as all sessions on the console terminal. **last** displays the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, **last** so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

**last reboot**

will give an indication of mean time between reboot.

**last** with no arguments displays a record of all logins and logouts, in reverse order.

If **last** is interrupted, it indicates how far the search has progressed in **wtmp.** If interrupted with a quit signal (generated by a CTRL-\) **last** indicates how far the search has progressed so far, and the search continues.

## OPTIONS

*–number*

limit the number of entries displayed to that specified by *number*.

**–f** *filename*

Use *filename* as the name of the accounting file instead of **/var/adm/wtmp.**

## FILES

**/var/adm/wtmp** login data base

## SEE ALSO

**lastcomm**(1), **utmp**(5), **ac**(8)

NAME

lastcomm – show the last commands executed, in reverse order

SYNOPSIS

**lastcomm** [ *command-name* ] ... [ *user-name* ] ... [ *terminal-name* ] ...

DESCRIPTION

**lastcomm** gives information on previously executed commands. **lastcomm** with no arguments displays information about all the commands recorded during the current accounting file's lifetime. If called with arguments, **lastcomm** only displays accounting entries with a matching *command-name*, *user-name*, or *terminal-name*.

EXAMPLES

The command:

**example% lastcomm a.out root ttyd0**

would produce a listing of all the executions of commands named **a.out**, by user **root** while using the terminal **ttyd0**. and

**example% lastcomm root**

would produce a listing of all the commands executed by user **root**.

For each process entry, **lastcomm** displays the following items of information:

- The command name under which the process was called.
- One or more flags indicating special information about the process. The flags have the following meanings:
  - **F** The process performed a **fork** but not an **exec**.
  - **S** The process ran as a set-user-id program.
  - **D** The process dumped memory.
  - **X** The process was killed by some signal.
- The name of the user who ran the process.
- The terminal which the user was logged in on at the time (if applicable).
- The amount of CPU time used by the process (in seconds).
- The date and time the process exited.

FILES

**/var/adm/acct** accounting file

SEE ALSO

**last**(1), **sigvec**(2), **acct**(5), **core**(5)

## NAME

ld, ld.so – link editor, dynamic link editor

## SYNOPSIS

**ld** [ **–align** *datum* ] [ **–assert** *assertion-keyword* ] [ **–A** *name* ] [ **–B***binding-keyword* ] [ **–d** ] [ **–dc** ] [ **–dp** ] [ **–D** *hex* ] [ **–e** *entry* ] [ **–l***x* ] [ **–L***dir* ] [ **–M** ] [ **–n** ] [ **–N** ] [ **–o** *name* ] [ **–p** ] [ **–r** ] [ **–s** ] [ **–S** ] [ **–t** ] [ **–T** [ **text** ] *hex* ] [ **–Tdata** *hex* ] [ **–u** *name* ] [ **–x** ] [ **–X** ] [ **–y***sym* ] [ **–z** ] *filename* ...

## DESCRIPTION

**ld** combines object programs to create an *executable* file or another object program suitable for further **ld** processing (with the **–r** option). The object modules on which **ld** operates are specified on the command line, and can be:

- simple object files, which typically end in the **.o** suffix, and are referred to as "dot-oh" files
- **ar**(1V) library archives (**.a**), or "libraries"
- dynamically-bound, sharable object files (**.so**), are also referred to as "shared libraries," which are created from previous **ld** executions.

Unless an output file is specified, **ld** produces a file named **a.out**. This file contains the object files given as input, appropriately combined to form an executable file.

## OPTIONS

When linking debugging or profiling objects, include the **–g** or **–pg** option (see **cc**(1V)), as appropriate, in the **ld** command.

Options should appear before filenames, except for abbreviated library names specified with **–l** options, and some binding control options specified by **–B** (which can appear anywhere in the line).

**–align** *datum*

Force the global uninitialized data symbol *datum* (usually a FORTRAN common block) to be page-aligned. Increase its size to a whole number of pages, and place its first byte at the start of a page.

**–assert** *assertion-keyword*

Check an assertion about the link editing being performed. The assertion desired is specified by the *assertion-keyword* string. **ld** is silent if the assertion holds, else it yields a diagnostic and aborts. Valid *assertion-keyword*'s and their interpretations are:

| | |
|---|---|
| **definitions** | If the resulting program were run now, there would be no run-time undefined symbol diagnostics. This assertion is set by default. |
| **nosymbolic** | There are no symbolic relocation items remaining to be resolved. |
| **pure-text** | The resulting load has no relocation items remaining in its text. |

**–A** *name*

Incremental loading: linking is to be done in a manner so that the resulting object may be read into an already executing program. *name* is the name of a file whose symbol table is taken as a basis on which to define additional symbols. Only newly linked material is entered into the text and data portions of **a.out**, but the new symbol table will reflect all symbols defined before and after the incremental load. This argument must appear before any other object file in the argument list. One or both of the **–T** options may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding addresses (which must be a multiple of the page size). The default value is the old value of **_end**.

**–B***binding-keyword*

Specify allowed binding times for the items which follow. Allowed values of *binding-keyword* are:

**dynamic** Allow dynamic binding: do not resolve symbolic references, allow creation of run-time symbol and relocation environment. **-Bdynamic** is the default. When **–Bdynamic** is in effect, all sharable objects encountered until a succeeding **–Bstatic** may be added dynamically to the object being linked. Non-sharable objects are bound statically.

**nosymbolic** Do not perform symbolic relocation, even if other options imply it.

**static** Bind statically. Opposite of **–Bdynamic**. Implied when either **–n** or **–N** is specified. Influences handling of all objects following its specification on a command line until the next **–Bdynamic**.

**symbolic** Force symbolic relocation. Normally implied if an entry point has been specified with **–e**, or if dynamic loading is in effect.

**–d** Force common storage for uninitialized variables and other common symbols to be allocated in the current **ld** run, even when the **–r** flag is present (which would otherwise postpone this binding until the final linking phase).

**–dc** Do **–d**, but also copy initialized data referenced by this program from shared objects.

**–dp** Force an alias definition of undefined procedure entry points. Used with dynamic binding to improve sharing and the locality of run-time relocations.

**–D** *hex* Pad the data segment with zero-valued bytes to make it *hex* bytes long.

**–e** *entry*

Define the entry point: the *entry* argument is made the name of the entry point of the loaded program. Implies **–Bsymbolic**.

**–l***x*[*.v*] This option is an abbreviation for the library name **lib***x***.a**, where *x* is a string. **ld** searches for libraries first in any directories specified with **–L** options, then in the standard directories **/lib**, **/usr/lib**, and **/usr/local/lib**. A library is searched when its name is encountered, so the placement of a **–l** is significant. If a dynamically loadable object is found, and **–Bdynamic** is in effect at that point on the command line, then **ld** prepares to access the object for relocation at run-time. In such a case, the optional *.v* suffix can be used to indicate a specific library version.

**–L***dir* Add *dir* to the list of directories in which to search for libraries. Directories specified with **–L** are searched before the standard directories, **/lib**, **/usr/lib**, and **/usr/local/lib**.

**–M** Produce a primitive load map, listing the names of the files which will be loaded.

**–n** Arrange (by giving the output file a 0410 "magic number") that when the output file is executed, the text portion will be read-only with the data areas placed at the beginning of the next address boundary following the end of the text. Implies **–Bstatic**.

**–N** Do not make the text portion read-only. (Use "magic number" 0407.) Implies **–Bstatic**.

**–o** *name*

*name* is made the name of the **ld** output file, instead of **a.out**.

**–p** Arrange for the data segment to begin on a page boundary, even if the text is not shared (with the **–N** option).

**–r** Generate relocation bits in the output file so that it can be the subject of another **ld** run. This flag also prevents final definitions from being given to common symbols, and suppresses the "undefined symbol" diagnostics.

**–s** Strip the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debuggers). This information can also be removed by **strip**(1).

**–S** Strip the output by removing all symbols except locals and globals.

**–t** Trace: display the name of each file as it is processed.

**–T [ text ]** *hex*
Start the text segment at location *hex*. Specifying **–T** is the same as using the **–Ttext** option.

**–Tdata** *hex*
Start the data segment at location *hex*. This option is only of use to programmers wishing to write code for PROMs, since the resulting code cannot be executed by the system.

**–u** *name*
Enter *name* as an undefined symbol. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

**–x** Preserve only global (non-**.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

**–X** Record local symbols, except for those whose names begin with **L**. This option is used by cc to discard internally generated labels while retaining symbols local to routines.

**–y***sym* Display each file in which *sym* appears, its type and whether the file defines or references it. Many such options may be given to trace many symbols. It is usually necessary to begin *sym* with an '_', as external C, FORTRAN and Pascal variables begin with underscores.

**–z** Arrange for the process demand paged from the resulting executable file (0413 "magic number"). This is the default. Results in a (32-byte) header on the output file followed by text and data segments, each of which has a multiple of page-size bytes (being padded out with NULL characters in the file if necessary). With this format the first few BSS segment symbols may actually end up in the data segment; this is to avoid wasting the space resulting from rounding the data segment size. Implies **–Bdynamic**.

## USAGE

### Command Line Processing

In general, options should appear ahead of the list of files to process. Unless otherwise specified, the effect of an option covers all of **ld** operations, independent of that option's placement on the command line. Exceptions to this rule include some of the binding control options specified by '**–B**' and the abbreviated library-names specified by '**–l**'. These may appear anywhere, and their influence is dependent upon their location. Some options may be obtained from environment variables, such options are interpreted before any on the command line (see ENVIRONMENT, below).

### Object File Processing

The files specified on the command line are processed in the order listed. Information is extracted from each file, and concatenated to form the output. The specific processing performed on a given file depends upon whether it is a simple object file, a library archive, or a shared library.

Simple object (**.o**) files are concatenated to the output as they are encountered.

Library archive (**.a**) files are searched exactly once each, as each is encountered; only those archive entries matching an unresolved external reference are extracted and concatenated to the output. If a member of an archive references a symbol defined by another member of that same archive, the member making the reference must appear before the member containing the definition.

On Sun386i, a library contains a dictionary of symbols, On other Sun systems, processing library archives through **ranlib**(1) provides this dictionary. In addition, you can use **lorder**(1), in combination with **tsort**(1) to place library members in calling order (see **lorder**(1) for details), or both (for fastest symbol lookup). The first member of an archived processed by **ranlib** has the reserved name of **__.SYMDEF**, which **ld** takes to be the dictionary of all symbols defined by members of the archive.

Sharable objects (**.so**) are scanned for symbol definitions and references, but are not normally included in the output from **ld**, except in cases where a shared library exports initialized data structures and the **–dc** option is in effect. However, the occurrence of each sharable object file in the **ld** command line is noted in the resulting executable file; this notation is utilized by an execution-time variant of **ld**, **ld.so**, for *deferred* and *dynamic* loading and binding during execution. See **Execution-Time Loading**, below, for details.

The **–l** option specifies a short name for an object file or archive used as a library. The full name of the object file is derived by adding the prefix **lib** and a suffix of either **.a** or **.so**[.*v*] to indicate an **ar**(1V) archive or a shared library, respectively. The specific suffix used is determined through rules discussed in **Binding and Relocation Semantics**, below.

**ld** searches for the desired object file through a list of directories specified by **–L** options, the environment variable **LD_LIBRARY_PATH**, and finally, the built-in list of standard library directories: **/lib**, **/usr/lib**, and **/usr/local/lib**.

**Binding and Relocation Semantics**

The manner in which **ld** processes a given object file is dependent in part upon the "binding mode" in which it is operating at the time the file is encountered. This binding mode is specified by the **–B** flag, which takes the keyword arguments:

**dynamic** Allow dynamic binding, do not resolve symbolic references, and allow creation of execution-time symbol and relocation information. This is the default setting.

**static** Force static binding, implied by options that generate non-sharable executable formats.

**–Bdynamic** and **–Bstatic** may be specified several times, and may be used to toggle each other on and off. Like **–l**, the influence of each depends upon its location within the command line. When **–Bdynamic** is in effect, **–l** searches may be satisfied by the first occurrence of either form of library (**.so** or **.a**), but if both are encountered, the **.so** form is preferred. When **–Bstatic** is in effect, **ld** refuses to use any **.so** libraries it encounters; it continue searching for the **.a** form. Furthermore, an explicit request to load a **.so** file is treated as an error.

After **ld** has processed all input files and command line options, the form of the output it produces is based on the information provided in both. **ld** first tries to reduce all symbolic references to relative numerical offsets within the executable it is building. To perform this "symbolic reduction," **ld** must be able to determine that:

- all information relating to the program has been provided, in particular, no **.so** is to be added at execution time; and/or
- the program has an entry point, and symbolic reduction can be performed for all symbols having definitions existing in the material provided.

It should be noted that uninitialized "common" areas (for example, uninitialized C globals) are allocated by the link editor *after* it has collected all references. In particular, this allocation can not occur in a program that still requires the addition of information contained in a **.so** file, as the missing information may affect the allocation process. Initialized "commons" however, are allocated within the executable in which their definition appears.

After **ld** has performed all the symbolic reductions it can, it attempts to transform all relative references to absolute addresses. **ld** is able to perform this "relative reduction" only if it has been provided *some* absolute address, either implicitly through the specification of an entry point, or explicitly through **ld** command-line options. If, after performing all the reductions it can, there are no further relocations or definitions to perform, then **ld** has produced a completely linked executable.

**Execution-Time Loading**

In the event that one or more reductions can not be completed, the executable will require further link editing at execution time in order to be usable. Such executables contain an data structure identified with the symbol **__DYNAMIC**. An incompletely linked "main" program should be linked with a "bootstrap" routine that invokes **ld.so**, which uses the information contained in the main program's **__DYNAMIC** to

assemble the rest of the executables constituting the entire program. A standard Sun compilation driver (such as **cc**(1V)) automatically includes such a module in each "main" executable.

When **ld.so** is given control on program startup, it finds all **.so** files specified when the program was constructed (and all **.so**'s on which they depend), and loads them into the address space. **ld.so** then completes all remaining relocations, with the exception of procedure call relocations; failure to resolve a given non-procedural relocation results in termination of the program with an appropriate diagnostic.

Procedure relocations are resolved when the referencing instruction is first executed. It should be noted that it is possible for "undefined symbol" diagnostics to be produced during program execution if a given target is not defined when referenced.

Although it is possible for binding errors to occur at execution-time, such an occurrence generally indicates something wrong in the maintenance of shared objects. **ld**'s **–assert definitions** function (on by default) checks at **ld**-time whether or not an execution-time binding error would occur.

**Version Handling for Shared Libraries**

To allow the independent evolution of **.so**'s used as libraries and the programs which use them, **ld**'s handling of **.so** files found through **–l** options involves the retention and management of version control information. The **.so** files used as such "shared libraries" are post-fixed with a Dewey-decimal format string describing the version of the "library" contained in the file.

The first decimal component is called the library's "major version" number, and the second component its "minor version" number. When **ld** records a **.so** used as a library, it also records these two numbers in the database used by **ld.so** at execution time. In turn, **ld.so** uses these numbers to decide which of multiple versions of a given library is "best" or whether *any* of the available versions are acceptable. The rules are:

- Major Versions Identical: the major version used at execution time must exactly match the version found at **ld**-time. Failure to find an instance of the library with a matching major version causes a diagnostic to be issued and the program's execution to be terminated.
- Highest Minor Version: in the presence of multiple instances of libraries that match the desired major version, **ld.so** uses the highest minor version it finds. However, if the highest minor version found at execution time is less than the version found at **ld**-time, a warning diagnostic is issued; program execution continues.

The semantics of version numbers are such that major version numbers should be changed whenever interfaces are changed. Minor versions should be changed to reflect compatible updates to libraries, and programs will silently favor the highest compatible version they can obtain.

**Special Symbols**

A number of symbols have special meanings to **ld** and programs should not define these symbols. The symbols described below are those actually seen by **ld**. Note: C and several other languages prepend symbols they use with '_'.

**_etext** The first location after the text of the program.

**_edata** The first location after initialized data.

**_end** The first location after all data.

**__DYNAMIC**
Identifies an **ld**-produced data structure. It is defined with a non-zero value in executables which require execution-time link editing. By convention, if defined, it is the first symbol in the symbol table associated with an **a.out** file.

**__GLOBAL_OFFSET_TABLE_**
A position-independent reference to an **ld**-constructed table of addresses. This table is constructed from "position-independent" data references occurring in objects that have been assembled with the assembler's **–k** flag (invoked on behalf of C compilations performed with the **–pic** flag). A related table (for which no symbol is currently defined) contains a series of transfer instructions and is created from "position-independent" procedure calls or, if **–dp** is specified to **ld**, a list of

undefined symbols.

Symbols in object files beginning with the letter **L** are taken to be local symbols and unless otherwise specified are purged from **ld** output files.

ENVIRONMENT

**LD_LIBRARY_PATH**

A colon-separated list of directories in which to search for libraries specified with the **–l** option. Similar to the **PATH** environment variable. **LD_LIBRARY_PATH** also affects library searching during execution-time loading.

**LD_OPTIONS**

A default set of options to **ld**. **LD_OPTIONS** is interpreted by **ld** just as though its value had been placed on the command line, immediately following the name used to invoke **ld**, as in:

**example% ld $LD_OPTIONS** ... *other-arguments* ...

Note: Environment variable-names beginning with the characters '**LD_**' are reserved for possible future enhancements to **ld**.

FILES

| | |
|---|---|
| **/usr/lib/lib*.a** | libraries |
| **lib*.so.v** | shared libraries |
| **lib*.sa.v** | exported, initialized shared library data |
| **/usr/lib/ld.so** | execution-time **ld** |
| **/usr/lib/*crt*.o** | default program bootstraps |
| **a.out** | output file |
| **/usr/local/lib** | |

SEE ALSO

as(1), **ar**(1V), **cc**(1V), **lorder**(1), **ranlib**(1), **strip**(1), **tsort**(1)

BUGS

Options are being overloaded and are an inappropriate vehicle for describing to **ld** the wide variety of things it can do. There needs to be a link-editing language which can be used in the more complex situations.

The **–r** option does not properly handle programs assembled with the **–k** (position-independent) flag, invoked from **cc** with **–pic** or **–PIC**.

**NAME**

ldd – list dynamic dependencies

**SYNOPSIS**

**ldd** *filename* ...

**DESCRIPTION**

For each *filename* , **ldd** lists the dynamically loaded objects on which that *filename* depends, if any. If the dynamic dependency is a "library" (a so-called "shared library"), then both the canonical form of the library name and version and the actual pathname used to access the library are listed. For example, if a given *filename* uses a shared C library version 4, which has the name **/usr/lib/libc.so.4.9** (where 9 is the most recent revision to interface version number 4) then **ldd** *filename* will report:

**filename:**

**-lc.4 => /usr/lib/libc.so.4.9**

For each *filename* which is not an executable program, or else does not require any dynamic objects for its execution, **ldd** will issue an appropriate diagnostic.

It should be noted that although all dynamically linked programs depend on the file **/usr/lib/ld.so , ldd** will never report this dependency.

**SEE ALSO**

**ld**(1)

NAME

leave – remind you when you have to leave

SYNOPSIS

**leave** [[ + ] *hhmm* ]

DESCRIPTION

**leave** sets an alarm to a time you specify and will tell you when the time is up. **leave** waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, and at the time. After the specified time, it reminds you every minute thereafter 10 more times. **leave** disappears after you log off.

You can specify the time in on of two ways, namely as an absolute time of day in the form *hhmm* where *hh* is a time in hours (on a 12 or 24 hour clock), or you can place a + sign in front of the time, in which case the time is relative to the current time, that is, the specified number of hours and minutes from now. All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, **leave** prompts with **'When do you have to leave?'**. **leave** exits if you just type a NEWLINE, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a **.login** or **.profile**.

**leave** ignores interrupts, quits, and terminates. To get rid of it you should either log off or use **kill –9** and its process ID.

EXAMPLES

The first example sets the alarm to an absolute time of day:

**example% leave 1535**
**Alarm set for Wed Mar 7 15:35:07 1984**

*work work work work*

**example% Time to leave!**

The second example sets the alarm for 10 minutes in the future:

**example% leave +10 Alarm set for Wed Mar 7**

*work work work work*

**example% Time to leave!**

*work work work work*

**example% You're going to be late!**

FILES

**.login**
**.profile**

SEE ALSO

**calendar**(1)

## NAME

lex – lexical analysis program generator

## SYNOPSIS

**lex** [ **–fntv** ] [ *filename* ] ...

## DESCRIPTION

**lex** generates programs to be used in simple lexical analysis of text. Each *filename* (the standard input by default) contains regular expressions to search for, and actions written in C to be executed when expressions are found.

A C source program, **lex.yy.c** is generated, to be compiled as follows:

**cc lex.yy.c –ll**

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized. The actual string matched is left in **yytext**, an external character array.

Matching is done in order of the strings in the file. The strings may contain square braces to indicate character classes, as in **[abx–z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *****, **+** and **?**, which mean, respectively, any nonnegative number, any positive number, or either zero or one occurrences of the previous character or character-class. The "dot" character (**.**) is the class of all ASCII characters except NEWLINE.

Parentheses for grouping and vertical bar for alternation are also supported. The notation $r\{d,e\}$ in a rule indicates instances of regular expression $r$ between $d$ and $e$. It has a higher precedence than |, but lower than that of *****, **?**, **+**, or concatenation. The **^** (carat character) at the beginning of an expression permits a successful match only immediately after a NEWLINE, and the **$** character at the end of an expression requires a trailing NEWLINE.

The / character in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, although the remainder of the expression must follow in the input stream.

An operator character may be used as an ordinary symbol if it is within "" symbols or preceded by '\'.

Three subroutines defined as macros are expected: **input**() to read a character; **unput**(*c*) to replace a character read; and **output**(*c*) to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex**(), and the library contains a **main**() which calls it. The action REJECT on the right side of the rule rejects this match and executes the next suitable match; the function **yymore**() accumulates additional characters into the same **yytext**; and the function **yyless**(*p*) pushes back the portion of the string matched beginning at *p*, which should be between **yytext** and **yytext+yyleng**. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

In a **lex** program, any line beginning with a blank is assumed to contain only C text and is copied; if it precedes **%%** it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a **%%**, as in YACC. Lines preceding **%%** which begin with a nonblank character define the string on the left to be the remainder of the line; it can be used later by surrounding it with **{}**. Note: curly brackets do not imply parentheses; only string substitution is done.

The external names generated by **lex** all begin with the prefix **yy** or **YY**.

Certain table sizes for the resulting finite-state machine can be set in the definitions section:

**%p** *n* number of positions is *n* (default 2000)

**%n** *n* number of states is *n* (500)

**%t** *n* number of parse tree nodes is *n* (1000)

**%a** *n* number of transitions is *n* (3000)

The use of one or more of the above automatically implies the –v option, unless the –n option is used.

OPTIONS

**–f** Faster compilation. Do not bother to pack the resulting tables; limited to small programs.

**–n** Opposite of –v; –n is default.

**–t** Place the result on the standard output instead of in file **lex.yy.c**.

**–v** Print a one-line summary of statistics of the generated analyzer.

EXAMPLES

The following command line:

**lex lexcommands**

would draw **lex** instructions from the file **lexcommands**, and place the output in **lex.yy.c**.

The following:

```
%% [A–Z]   putchar (yytext[0]+'a'–'A'); [ ]+$     ; [ ]+      putchar(' ');
```

is an example of a **lex** program. It converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
D       [0–9]
%%
if      printf("IF statement\n");
[a–z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input( ) != '*');
                switch (input( ))
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: go to loop;
                        }
                }
```

FILES

**lex.yy.c**

SEE ALSO

**sed**(1V), **yacc**(1)

*Programming Utilities and Libraries*

NAME

line – read one line

SYNOPSIS

**line**

DESCRIPTION

**line** copies one line (up to a NEWLINE) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints a NEWLINE at least. It is often used within shell scripts to read a line from the terminal.

SEE ALSO

**sh**(1), **read**(2V)

## NAME

lint – a C program verifier

## SYNOPSIS

**lint** [ **–abchinquvxz** ] [ **–D***name* [*=def*] ] [ **–host=***arch* ] [ **–I***directory* ] [ **–l***library* ] [ **–o** *outputfile* ]
[ **–target=***arch* ] [ **–U***name* ] *filename* ...

**lint** [ **–C***library* ] [ **–D***name* [*=def*] ] [ **–host=***arch* ] [ **–I***directory* ] [ **–l***library* ] [ **–target=***arch* ]
[ **–U***name* ] *filename* ...

## SYSTEM V SYNOPSIS

**/usr/5bin/lint** [ **–abcghnpquvxzO** ] [ **–D***name* [*=def*] ] [ **–I***directory* ] [ **–l***library* ] [ **–o** *outputfile* ]
[ **–U***name* ] *filename* ...

## DESCRIPTION

**lint** attempts to detect features of the named C program files that are likely to be bugs, to be non-portable, or to be wasteful. It also performs stricter type checking than does the C compiler. **lint** runs the C preprocessor as its first phase, with the preprocessor symbol **lint** defined to allow certain questionable code to be altered or skipped by **lint**. Therefore, this symbol should be thought of as a reserved word for all code that is to be checked by **lint**.

Among the possible problems that are currently noted are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions with constant values. Function calls are checked for inconsistencies, such as calls to functions that return values in some places and not in others, functions called with varying numbers of arguments, function calls that pass arguments of a type other than the type the function expects to receive, functions whose values are not used, and calls to functions not returning values that use the non-existent return value of the function.

Filename arguments ending with **.c** are taken to be C source files. Filename arguments with names ending with **.ln** are taken to be the result of an earlier invocation of **lint**, with either the **–i** or the **–C** option in effect. The **.ln** files are analogous to the **.o** (object) files produced by the **cc**(1V) from **.c** files. **lint** also accepts special libraries specified with the **–l** option, which contain simplified definitions of standard library routines (preprocessed by '**lint –C**') for faster checking of function calls.

**lint** processes the various **.c**, **.ln**, and **llib-l***library***.ln** (lint library) files and process them in command-line order. By default, **lint** appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. When the **–C** and **–i** options are *omitted* the second pass of **lint** checks this list of files for mutual compatibility. When the **–C** or the **–i** options are used, the **.ln** and the **llib-l***library***.ln** files are ignored.

## SYSTEM V DESCRIPTION

Filename arguments with names ending with **.ln** are taken to be the result of an earlier invocation of **lint**, with either the **–c** or the **–o** option in effect.

**lint** processes the various **.c**, **.ln**, and **llib-l***library***.ln** (lint library) files and process them in command-line order. By default, **lint** appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is *omitted* the second pass of **lint** checks this list of files for mutual compatibility. When the **–c** option is used, the **.ln** and the **llib-l***library***.ln** files are ignored.

**lint** produces its first-pass output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. If the **–c** option is not used, information gathered from all input files is then collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name is printed, followed by a question mark.

## OPTIONS

**–a** Report assignments of **long** values to variables that are not **long**.

**–b** Report **break** statements that cannot be reached. This is not the default because, unfortunately, most **lex**(1) and many **yacc**(1) outputs produce many such complaints.

**–c** Complain about casts which have questionable portability.

**–h** Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

**–i** Produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of **lint**'s first pass only, and are not checked for compatibility between functions.

**–n** Do not check compatibility against the standard library.

**–q** Do not complain about constructs that do not cause portability problems between current Sun implementations of the C language but that will cause portability problems between other implementations. If the **–q** flag is specified, **lint** treats type **enum** as an **int**, treats type **long** as type **int** and type **unsigned long** as **unsigned int**, and treats a 0 argument as being conformable with any pointer.

**–u** Do not complain about functions and external variables used and not defined, or defined and not used (this is suitable for running **lint** on a subset of files comprising part of a larger program).

**–v** Suppress complaints about unused arguments in functions.

**–x** Report variables referred to by **extern** declarations, but never used.

**–z** Do not complain about structures that are never defined (for example, using a structure pointer without knowing its contents).

**–C***library*
Create a **lint** library with the name **llib-l***library***.ln**.

**–D***name*[*=def*]
Define *name* for **cpp**(1), as if by a **#define** directive. If no definition is given, *name* is defined as 1.

**–host=***arch* (Sun-2, Sun-3, and Sun-4 systems only)
Define the host architecture to be *arch*. The default is the architecture returned by the **arch**(1) command. *arch* can be one of **sun2**, **sun3**, or **sun4**.

**–I***directory*
Add *directory* to the list of list of directories in which to search for include files. Include files with names that do not begin with / are always sought first in the directory of the *filename* argument, then in directories named in **–I** options, then in the **/usr/include** directory.

**–l***library*
Use the lint library *library* from the **/usr/lib/lint** directory.

**–o** *outputfile*
Name the output file *outputfile*. *outputfile* cannot be the same as *sourcefile* (**lint** will not overwrite the source file).

**–target=***arch* (Sun-2, Sun-3, and Sun-4 systems only)
Define the target architecture to be *arch*, for additional portability checks specific to that architecture. The default is the value returned by the **arch**(1) command. *arch* can be one of: **sun2**, **sun3**, or **sun4**.

**–U***name*
Remove any initial definition of *name* for the preprocessor.

SYSTEM V OPTIONS

The sense of the **–a**, **–b**, **–h**, and **–x** options is reversed in the System V version of **lint**; the tests they control *are* performed unless the flag is specified. The **–C** option is not available; instead, the **–c** or **–o** options can be used. The **–i** option is not used; instead, the **–c** option can be used. The **–q**, **–host**, and **–target** options are not available.

**–c** Produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of **lint**'s first pass only, and are not checked for compatibility between functions.

**-g**

**-O** These options are accepted but ignored. By recognizing these options, **lint**'s behavior is closer to that of the **cc**(1V) command.

**-n** Do not check compatibility against either the standard or the portable lint library.

**-p** Attempt to check portability of code to other dialects of C, such as IBM 370 and Honeywell GCOS. Along with performing stricter checking, this option truncates all non-external names to eight characters, and all external names to six characters and one case.

**-o** *library*

Create a lint library with the name **llib-l***lib***.ln**. The **-c** option nullifies any use of the **-o** option. The lint library produced is the input that is given to **lint**'s second pass. The **-o** option simply saves this file in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **-x** option is suggested. The **-v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see **Input Grammar** below).

USAGE

For more information about **lint** refer to **lint** in *Programming Utilities and Libraries*

To create lint libraries, use the **-C** option. For example

**example% lint -Ccongress** *filename*s ...

where *filename*s are the C sources of library *congress*, produces a file **llib-lcongress.ln** in the current directory in the correct library format suitable for "linting" programs using **-lcongress.**

**Input Grammar**

**lint**'s first pass reads standard C source files. **lint** recognizes the following C comments as commands.

**/*NOTREACHED*/**

At appropriate points, inhibit comments about unreachable code. (This comment is typically placed just after calls to functions like **exit**(2)).

**/*VARARGS*n*/**

Suppress the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0. In this version of **lint, /*VARARGS0*/** is allowed. It no longer indicates the absence of variable arguments.

**/*ARGSUSED*/**

Enable the **-v** option for the next function.

**/*LINTLIBRARY*/**

At the beginning of a file, shut off complaints about unused functions and function arguments in this file. This is equivalent to using the **-v** and **-x** options.

SYSTEM V USAGE

The behavior of the **-c** and the **-o** options allows for incremental use of **lint** on a set of C source files. Invoking '**lint -c**' for each source file produces a corresponding **.ln** file, and prints all messages pertaining to that source file. After all of the source files have been run through **lint** separately, it is invoked once more (without the **-c** option), and with all of the **.ln** files and **-l***x* options. This produces messages about any inconsistencies between files. This scheme works well with **make**(1), since it allows **make** to "lint" only those source files that have been modified since the last time **lint** was run.

To create lint libraries, use the **-o** option. For example

**example% lint -x -o congress** *filename*s ...

where *filename*s are the C sources of library *congress*, produces a file **llib-lcongress.ln** in the current directory in the correct library format suitable for "linting" programs using **-lcongress.**

## EXAMPLE

The following **lint** call:

**example% lint -b myfile.c**

checks the consistency of the code in the C source file file **myfile.c**. The **-b** option indicates that unreachable **break** statements are to be checked for.

## FILES

| | |
|---|---|
| **/usr/lib/lint/lint[12]** | programs |
| **/usr/lib/lint/llib-l*.ln** | various prebuilt lint libraries |
| **/usr/lib/lint/llib-l*** | sources of the prebuilt lint libraries |

The following lint libraries are supplied with SunOS: **-lc, -lcore, -lcurses, -lkvm, -llwp, -lm, -lmp, -lpixrect, -lplot, -lsuntool, -lsunwindow, -ltermcap,** and **-ltermlib**. Additional lint libraries may be installed separately.

## SYSTEM V FILES

| | |
|---|---|
| **/usr/5lib/lint/lint[12]** | programs |
| **/usr/5lib/lint/llib-l*.ln** | various prebuilt lint libraries |
| **/usr/5lib/lint/llib-l*** | sources of the prebuilt lint libraries |

The following System V lint libraries are supplied with SunOS: **-lc, -lcore, -lcurses, -lkvm, -llwp, -lm, -lmp, -lpixrect, -lplot, -lport. -lsuntool,** and **-lsunwindow**. Additional lint libraries may be installed separately.

## SEE ALSO

**cc**(1V), **cpp**(1), **lex**(1), **make**(1), **yacc**(1), **exit**(2), **setjmp**(3)

*Programming Utilities and Libraries*

## BUGS

There are some things you just *cannot* get **lint** to shut up about.

The routines **exit**(2), **longjmp** (see **setjmp**(3)) and other functions that do not return are not understood; this causes various incorrect diagnostics.

Libraries created by the **-C** or **-o** options will, when used in later **lint** runs, cause certain errors that were reported when the libraries were created to be reported again, and cause line numbers and file names from the original source used to create those libraries to be reported in error messages. For these reasons, it is still useful to produce stripped down lint library source files and to use them to generate lint libraries.

NAME

ln – make hard or symbolic links to files

SYNOPSIS

**ln** [ **–fs** ] *filename* [ *linkname* ]
**ln** [ **–fs** ] *pathname... directory*

DESCRIPTION

**ln** creates an additional directory entry, called a *link*, to a file or directory. Any number of links can be assigned to a file. The number of links does not affect other file attributes such as size, protections, data, etc.

*filename* is the name of the original file or directory. *linkname* is the new name to associate with the file or filename. If *linkname* is omitted, the last component of *filename* is used as the name of the link.

If the last argument is the name of a directory, symbolic links are made in that directory for each *pathname* argument; **ln** uses the last component of each *pathname* as the name of each link in the named *directory*.

A hard link (the default) is a standard directory entry just like the one made when the file was created. Hard links can only be made to existing files. Hard links cannot be made across file systems (disk partitions, mounted file systems). To remove a file, all hard links to it must be removed, including the name by which it was first created; removing the last hard link releases the inode associated with the file.

A symbolic link, made with the **–s** option, is a special directory entry that points to another named file. Symbolic links can span file systems and point to directories. In fact, you can create a symbolic link that points to a file that is currently absent from the file system; removing the file that it points to does not affect or alter the symbolic link itself.

A symbolic link to a directory behaves differently than you might expect in certain cases. While an **ls**(1V) on such a link displays the files in the pointed-to directory, an '**ls –l**' displays information about the link itself:

```
example% ln –s dir link
example% ls link
file1 file2 file3 file4
example% ls –l link
lrwxrwxrwx  1 user        7 Jan 11 23:27 link -> dir
```

When you **cd**(1) to a directory through a symbolic link, you wind up in the pointed-to location within the file system. This means that the parent of the new working directory is not the parent of the symbolic link, but rather, the parent of the pointed-to directory. For instance, in the following case the final working directory is **/usr** and not **/home/user/linktest**.

```
example% pwd
/home/user/linktest
example% ln –s /usr/tmp symlink
example% cd symlink
example% cd ..
example% pwd
/usr
```

C shell user's can avoid any resulting navigation problems by using the **pushd** and **popd** built-in commands instead of **cd**.

OPTIONS

**–f** Force a hard link to a directory — this option is only available to the super-user.

**–s** Create a symbolic link or links.

EXAMPLE

The commands below illustrate the effects of the different forms of the **ln** command:

```
example% ln file link
example% ls –F file link
file  link
example% ln –s file symlink
example% ls –F file symlink
file  symlink@
example% ls –li file link symlink
 10606 -rw-r--r--  2 user         0 Jan 12 00:06 file
 10606 -rw-r--r--  2 user         0 Jan 12 00:06 link
 10607 lrwxrwxrwx  1 user         4 Jan 12 00:06 symlink -> file
example% ln –s nonesuch devoid
example% ls –F devoid
devoid@
example% cat devoid
devoid: No such file or directory
example% ln –s /proto/bin/* /tmp/bin
example% ls –F /proto/bin /tmp/bin
/proto/bin:
x*    y*    z*

/tmp/bin:
x@    y@    z@
```

SEE ALSO

**cp**(1), **ls**(1V), **mv**(1), **rm**(1), **link**(2), **lstat**(2), **readlink**(2), **stat**(2), **symlink**(2)

WARNINGS

When the last argument is a directory, simple basenames should not be used for *pathname* arguments. If a basename is used, the resulting symbolic link points to itself:

```
example% ln –s file /tmp
example% ls –l /tmp/file
lrwxrwxrwx  1 user         4 Jan 12 00:16 /tmp/file -> file
example% cat /tmp/file
/tmp/file: Too many levels of symbolic links
```

To avoid this problem, use full pathnames, or prepend a reference to the **PWD** variable to files in the working directory:

```
example% rm /tmp/file
example% ln –s $PWD/file /tmp
lrwxrwxrwx  1 user         4 Jan 12 00:16 /tmp/file -> /home/user/subdir/file
```

NAME

load, loadc – load Application SunOS or Developer's Toolkit clusters

SYNOPSIS

**load** [ *filename* ... ]

**loadc** [ *cluster* ... ]

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**load** loads the optional clusters in the Application SunOS or the Developer's Toolkit that contain the files specified in the filename arguments. **loadc** loads the optional clusters in the Application SunOS or the Developers Toolkit specified in the cluster arguments.

**load** and **loadc** require the user to specify the distribution media type (3.5" diskette or 1/4" tape) for the system and to insert the specified 3.5" diskette or 1/4" tape. The user will be asked to confirm that the specified media has been inserted. If the user confirmation is negative, no software will be loaded from the specified media.

Without arguments, **load** and **loadc** display a summary of the clusters in the Application SunOS and Developer's Toolkit, including the load state and size of each cluster.

EXAMPLES

To load the cluster that contains the **spell**(1) command:

```
% load spell
Enter your distribution media type (1=1/4" tape, 2=3.5" diskette): 2
Insert diskette n to load the spellcheck cluster, confirm (y/n): y
Loading the spellcheck cluster ...
The spellcheck cluster has been loaded.
space used by clusters: 6021K bytes
total space remaining: 20432K bytes
```

To load the **spellcheck** cluster:

```
% loadc spellcheck
Enter your distribution media type (1=1/4" tape, 2=3.5" diskette): 2
Insert diskette n to load the spellcheck, confirm (y/n): y
Loading the spellcheck cluster ...
The spellcheck cluster has been loaded.
space used by clusters: 6021K bytes
total space remaining: 20432K bytes
```

To display a summary of the clusters in the Application SunOS and Developer's Toolkit:

```
% load
Application SunOS Clusters:
  availablecluster          size (bytes)
  ---------  -------        ----
   yes       accounting     265K
   no        advanced_admin 501K
   ...
```

**Developer's Toolkit Clusters:**

| available | cluster | size (bytes) |
|---|---|---|
| no | base_devel | 6907K |
| ... | | |

**space used by clusters: 6021K bytes**
**total space remaining: 20432K bytes**

A cluster is available if it has been "loaded" using **load** or **loadc** or if it has been "mounted" across the network.

ENVIRONMENT

LOADMEDIA Used to specify the distribution media type for the system. It can be set to **diskette** to specify 3.5" diskette or **tape** to specify 1/4" tape. If it is set, **load** and **loadc** will not ask the user to enter the distribution media type.

FILES

| | |
|---|---|
| **/export/loaded/appl** | where Application SunOS clusters are loaded (or mounted) |
| **/export/loaded/devel** | where Developer's Toolkit clusters are loaded (or mounted) |
| **/usr/lib/load/*** | data files |

SEE ALSO

**unload**(1), **cluster(1)**, **toc**(5)

*Sun386i System Setup and Maintenance*

DIAGNOSTICS

**Wrong** *diskette/tape*
An incorrect diskette or tape was inserted. The user will again be asked to insert the specified media.

**The file** *filename* **is not in any of the optional software clusters.**
The specified file is not part of the Application SunOS or Developer's Toolkit.

**There is no** *cluster* **cluster.**
The specified cluster is not part of the Application SunOS or Developers Toolkit.

**The cluster** *cluster* **is already loaded, overwrite? (y/n):**
The specified cluster appears to have been loaded already. Type **y** followed by RETURN to have the cluster loaded or **n** followed by RETURN to cancel the loading of the cluster.

**Cluster** *cluster* **requires** *n***K; there is not enough disk space.**
There is not enough disk space to hold the specified cluster.

**The** *cluster* **cluster has not been loaded.**
The loading of the specified cluster has been canceled or interrupted by the user.

**The Application SunOS (and/or) Developers Toolkit are mounted.**
The Application SunOS or Developers Toolkit or both are mounted across the network and can not be loaded or unloaded.

**The** *tape/diskette* **drive is currently in use.**
You are trying to load a cluster from tape (or diskette) and another process currently has control of the tape (or diskette) drive.

NAME

lockscreen, lockscreen_default – maintain SunView context and prevent unauthorized access

SYNOPSIS

**lockscreen** [ **–enr** ] [ **–b** *program* ] [ **–t** *seconds* ] [ *gfx-program* [ *gfx-program-arguments* ] ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**lockscreen** is a SunView utility that locks the screen against unauthorized access while preserving the state of the SunView display. It clears the workstation screen to black, and then runs *gfx-program*, which typically provides a moving graphics display to reduce phosphor burn. If no *gfx-program* is provided, a suitable default program is run.

**lockscreen** requires the user's password before restoring the window context. When any keyboard or mouse button is pressed, the graphics screen is replaced by a password screen that displays the user name, a small box with a bouncing logo, and a prompt for the user's password. If the user has no password, or if the **–n** option is used, the window context is simply restored.

When the password screen appears:

- Restore the window context by entering the user's password followed by a RETURN (this password is not echoed on the screen) or,
- Point to the black box and click the left button to return to the graphics display.

If neither of the above actions is taken, *gfx_program* will resume execution after the interval specified with the **–t** option.

OPTIONS

**–e** Add the **Exit Desktop** choice to the password screen. If pointed to and clicked, the SunView environment is exited and the current user is logged out.

**–n** Require no password to reenter the window environment.

**–r** Allow the use of the user name **root** in the **'Name:'** field of the password screen. Normally, **root** is not accepted as a valid user name.

**–b** *program*

Allow an additional program to be run as a child process of **lockscreen**. This background process could be a compile server or some other useful program that the user wants run while lockscreen is running. No arguments are passed to this program.

**–t** *seconds*

After *seconds* seconds, clear the password screen and restart *gfx-program*. The default is 5 minutes (300 seconds).

[ *gfx-program* ] [ *gfx-program-arguments* ]

Run this program after clearing the screen to black. If no *program* argument is present, **lockscreen** will try to run **lockscreen_default** if it exists on the standard search path, otherwise a bouncing Sun logo will appear. If *gfx-program-arguments* are specified and the *gfx-program* is not then the arguments are passed to **lockscreen_default**. **lockscreen_default** is typically a non-interactive graphics program (see **graphics_demos**(6)). **lockscreen** will not search for **lockscreen_default** if the *gfx-program* is specified explicitly as an empty argument (an adjacent pair of quote-marks).

FILES

**/usr/bin/lockscreen_default**

Default *gfx-program*. This displays a series of **life**(6) patterns. If a file named **lockscreen_default** appears earlier in the search path, that file is used instead.

**SEE ALSO**

**login**(1), **sunview**(1)

NAME

logger – add entries to the system log

SYNOPSIS

**logger** [ **–t** *tag* ] [ **–p** *priority* ] [ **–i** ] [ **–f** *filename* ] [ *message* ] ...

DESCRIPTION

**logger** provides a method for adding one-line entries to the system log file from the command line. One or more *message* arguments can be given on the command line, in which case each is logged immediately. Otherwise, a *filename* can be specified, in which case each line in the file is logged. If neither is specified, **logger** reads and logs messages on a line-by-line basis from the standard input.

OPTIONS

| | |
|---|---|
| **–t** *tag* | Mark each line added to the log with the specified *tag*. |
| **–p** *priority* | Enter the message with the specified *priority*. The message priority can be specified numerically, or as a *facility.level* pair. For example, '**–p local3.info**' assigns the message priority to the **info** level in the **local3** facility. The default priority is **user.notice**. |
| **–i** | Log the process ID of the **logger** process with each line. |
| **–f** *filename* | Use the contents of *filename* as the message to log. |
| *message* | If this is unspecified, either the file indicated with **–f** or the standard input is added to the log. |

EXAMPLES

**logger System rebooted**

will log the message '**System rebooted**' to the facility at priority **notice** to be treated by **syslogd** as other messages to the facility **notice** are.

**logger –p local0.notice –t HOSTIDM –f /dev/idmc**

will read from the file **/dev/idmc** and will log each line in that file as a message with the tag '**HOSTIDM**' at priority **notice** to be treated by **syslogd** as other messages to the facility **local0** are.

SEE ALSO

**syslog**(3), **syslogd**(8)

NAME

login – log in to the system

SYNOPSIS

**login** [ **–p** ] [ *username* ]

DESCRIPTION

**login** signs *username* on to the system initially; **login** may also be used at any time to change from one userID to another.

When used with no argument, **login** requests a user name and password (if appropriate). Echoing is turned off (if possible) while typing the password. Note: the number of significant characters in a password is 8. (See **passwd**(1).)

When successful, **login** updates accounting files, prints the message of the day, informs you of the existence of any mail, and displays the time you last logged in. None of these messages are printed if there is a **.hushlogin** file in your home directory; this is mostly used to make life easier for nonhuman users, such as **uucp**(1C).

**login** initializes the user and group IDs and the working directory, then starts a command interpreter shell (usually either **/usr/bin/sh** or **/usr/bin/csh**) according to specifications found in the file **/etc/passwd**. Argument 0 of the command interpreter is the name of the command interpreter with a leading dash ('–') prepended.

**login** also modifies the environment (**environ**(5V)) with information specifying home directory, command interpreter, terminal-type (if available) and username. The **–p** argument preserves the remainder of the environment, otherwise any previous environment is discarded.

The super-user **root** may only log in on those terminals marked as "secure" in the **/etc/ttytab** file. For example, if the file contained:

```
console  "/etc/getty Console-9600"      sun     on secure
tty00    "/etc/getty Console-9600"      sun     on
...
```

the super-user could only log in on the console.

If the file **/etc/nologin** exists, **login** prints its contents on the user's terminal and exits. This is used by **shutdown**(8) to stop logins when the system is about to go down.

The **login** command, recognized by **sh**(1) and **csh**(1), is executed directly (without forking), and terminates that shell. To resume working, you must log in again.

**login** times out and exits if its prompt for input is not answered within a reasonable time.

When the Bourne shell (**sh**) starts up, it reads a file called **.profile** from your home directory (that of the username you use to log in). When the C shell (**csh**) starts up, it reads a file called **.cshrc** from your home directory, and then reads a file called **.login**.

The shells read these files only if they are owned by the person logging in.

OPTIONS

**–p** Preserve any existing environment variables and their values; otherwise the previous environment is discarded.

FILES

| | |
|---|---|
| **/etc/utmp** | accounting |
| **/var/adm/wtmp** | accounting |
| **/var/adm/lastlog** | time of last login |
| **/etc/ttytab** | terminal types |
| **/usr/ucb/quota** | quota check |
| **/var/spool/mail/*** | mail |
| **/etc/motd** | message-of-the-day |

| | |
|---|---|
| **/etc/passwd** | password file |
| **/etc/nologin** | stop login, print message |
| **/.hushlogin** | makes login quieter |
| **/usr/bin/sh** | |
| **/usr/bin/csh** | |
| **.login** | |
| **.profile** | |

SEE ALSO

**csh**(1), **mail**(1), **passwd**(1), **rlogin**(1C) **sh**(1), **uucp**(1C), **passwd**(5), **environ**(5V), **utmp**(5), **init**(8), **getty**(8), **shutdown**(8),

DIAGNOSTICS

**Login incorrect**

If the name or the password is bad (or mistyped).

**No Shell**

**cannot open password file**

**no directory**

Ask your system administrator for assistance.

**NAME**

**logname** – get the name by which you logged in

**SYNOPSIS**

**logname**

**DESCRIPTION**

**logname** returns the contents of the environment variable LOGNAME, which is set when a user logs into the system.

**FILES**

**/etc/profile**

**SEE ALSO**

**env**(1), **login**(1), **environ**(5V)

**NAME**

look – find words in the system dictionary or lines in a sorted list

**SYNOPSIS**

**look** [ **–df** ] [ –t*c* ] *string* [ *filename* ]

**DESCRIPTION**

**look** consults a sorted *filename*
and prints all lines that begin with *string*.

If no *filename* is specified, **look** uses **/usr/dict/words** with collating sequence **–df**.

**OPTIONS**

**–d** Dictionary order. Only letters, digits, TAB and SPACE characters are used in comparisons.

**–f** Fold case. Upper case letters aren't distinguished from lower case in comparisons.

–t*c* Set termination character. All characters to the right of *c* in *string* are ignored.

**FILES**

**/usr/dict/words**

**SEE ALSO**

**grep(1V), sort(1V)**

NAME

lookbib – find references in a bibliographic database

SYNOPSIS

**lookbib** *database*

AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '**%**', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '**%**'.

**lookbib** uses an inverted index made by **indxbib** to find sets of bibliographic references. It reads keywords typed after the '**>**' prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another '**>**' prompt.

It is possible to search multiple databases, as long as they have a common index made by **indxbib**(1). In that case, only the first argument given to **indxbib** is specified to **lookbib.**

If **lookbib** does not find the index files (the **.i[abc]** files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a **.ig** suffix, suitable for use with **fgrep** (see **grep**(1V)). **lookbib** then uses this **fgrep** file to find references. This method is simpler to use, but the **.ig** file is slower to use than the **.i[abc]** files, and does not allow the use of multiple reference files.

FILES

| | |
|---|---|
| ***x*.ia** | |
| ***x*.ib** | |
| ***x*.ic** | index files |
| *x*.ig | reference file |

SEE ALSO

**addbib**(1), **grep**(1V), **indxbib**(1), **refer**(1), **roffbib**(1), **sortbib**(1)

*Formatting Documents*

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

## NAME

lorder – find an ordering relation for an object library

## SYNOPSIS

**lorder** *filename...*

## DESCRIPTION

Give **lorder** one or more object or library archive (see **ar**(1V)) *filename*s, and it lists pairs of object file names — meaning that the first file of the pair refers to external identifiers defined in the second — to the standard output. **lorder**'s output may be processed by **tsort**(1) to find an ordering of a library suitable for one-pass access by **ld**(1).

## EXAMPLE

This brash one-liner intends to build a new library from existing **.o** files.

```
ar cr library `lorder *.o | tsort`
```

The **ranlib**(1), command converts an ordered archive into a randomly accessed library and makes **lorder** unnecessary.

## SEE ALSO

**ar**(1V), **ld**(1), **ranlib**(1), **tsort**(1)

## BUGS

The names of object files, in and out of libraries, must end with '**.o**'; otherwise, nonsense results.

NAME

lpq – display the queue of printer jobs

SYNOPSIS

**lpq** [ –P*printer* ] [ –**l** ] [ + [ *interval* ] ] [ *job#* ... ] [ *username* ... ]

DESCRIPTION

**lpq** displays the contents of a printer queue. It reports the status of jobs specified by *job#*, or all jobs owned by the user specified by *username*. **lpq** reports on all jobs in the default printer queue when invoked with no arguments.

For each print job in the queue, **lpq** reports the user's name, current position, the names of input files comprising the job, the job number (by which it is referred to when using **lprm**(1)) and the total size in bytes. Normally, only as much information as will fit on one line is displayed. Jobs are normally queued on a first-in-first-out basis. Filenames comprising a job may be unavailable, such as when **lpr** is used at the end of a pipeline; in such cases the filename field indicates "(standard input)".

If **lpq** warns that there is no daemon present (that is, due to some malfunction), the **lpc**(8) command can be used to restart a printer daemon.

OPTIONS

–**P** *printer*

Display information about the queue for the specified *printer*. In the absence of the –**P** option, the queue to the printer specified by the PRINTER variable in the environment is used. If the PRINTER variable isn't set, the queue for the default printer is used.

–**l** Display queue information in long format; includes the name of the host from which the job originated.

+[*interval*]

Display the spool queue periodically until it empties. This option clears the terminal screen before reporting on the queue. If an *interval* is supplied, **lpq** sleeps that number of seconds in between reports.

FILES

| | |
|---|---|
| **/etc/termcap** | for manipulating the screen for repeated display **/etc/printcap** to determine printer characteristics **/var/spool/l*** spooling directory, as determined from printcap **/var/spool/l*/cf*** control files specifying jobs **/var/spool/l*/lock** lock file to obtain the currently active job |

SEE ALSO

**lpr**(1), **lprm**(1), **lpc**(8), **lpd**(8)

DIAGNOSTICS

*printer* **is ready and printing**

The **lpq** program checks to see if there is a printer daemon. If the daemon is hung, the super-user can abort the current daemon and start a new one using **lpc**(8). Under some circumstances, **lpq** reports that a printer is ready and printing when the daemon is, in fact, hung.

**Waiting for** *printer* **to become ready (offline ?)**

The daemon could not open the printer device. The printer may be turned off-line. This message can also occur if a printer is out of paper, the paper is jammed, and so on. Another possible cause is that a process, such as an output filter, has exclusive use of the device. The only recourse in this case is to kill the offending process and restart the printer with *lpc*.

**waiting for** *host* **to come up**

A daemon is trying to connect to the remote machine named *host*, in order to send the files in the local queue. If the remote machine is up, **lpd** on the remote machine is probably dead or hung and should be restarted using *lpc*.

**sending to** *host*

The files are being transferred to the remote *host*, or else the local daemon has hung while trying to transfer the files.

**Warning:** *printer* **is down**

The printer has been marked as being unavailable with **lpc**.

**Warning: no daemon present**

The **lpd** process overseeing the spooling queue, as indicated in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. Check the printer's error log for a diagnostic from the deceased process; you can restart the printer daemon with **lpc**.

BUGS

**lpq** may report unreliably. The status as reported may not always reflect the actual state of the printer.

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

**lpq** is sometimes unable to open various files when the lock file is malformed.

NAME

lpr – send a job to the printer

SYNOPSIS

**lpr** [ **–P***printer* ] [ **–#***copies* ] [ **–C***class* ] [ **–J***job* ] [ **–T***title* ] [ **–i** [ *indent* ] ] [ **–1234***font* ]
[ **–w***cols* ] [ **–B** ] [ **–r** ] [ **–m** ] [ **–h** ] [ **–s** ] [ *–filter-option* ] [ *filename* ... ]

DESCRIPTION

**lpr** forwards printer jobs to a spooling area for subsequent printing as facilities become available. Each printer job consists of copies of (or, with **–s** , symbolic links to) each *filename* you specify. The spool area is managed by the line printer daemon, **lpd**(8). **lpr** reads from the standard input if no files are specified.

OPTIONS

**–P***printer*

Send output to the named *printer*. Otherwise send output to the printer named in the **PRINTER** environment variable, or to the default printer, **lp**.

**–#***copies*

Produce the number of *copies* indicated for each named file. For example:

**example% lpr –#3 index.c lookup.c**

produces three copies of **index.c**, followed by three copies of **lookup.c**. On the other hand,

**example% cat index.c lookup.c | lpr**

generates three copies of the concatenation of the files.

**–C***class*

Print *class* as the job classification on the burst page. For example,

**example% lpr –C Operations new.index.c**

replaces the system name (the name returned by *hostname*) with "Operations" on the burst page, and prints the file *new.index.c*.

**–J***job* Print *job* as the job name on the burst page. Normally, **lpr** uses the first file's name.

**–T** *title* Use *title* instead of the file name for the title used by **pr**(1V).

**–i***[indent]*

Indent output *indent* spaces. Eight spaces is the default.

**–1***font*
**–2***font*
**–3***font*
**–4***font* Mount the specified *font* on font position **1**, **2**, **3** or **4**. The daemon will construct a *.railmag* file in the spool directory that indicates the mount by referencing **/usr/lib/vfont/font**.

**–w***cols* Use *cols* as the page width for **pr**.

**–r** Remove the file upon completion of spooling, or upon completion of printing with the **–s** option.

**–m** Send mail upon completion.

**–h** Suppress printing the burst page.

**–s** Create a symbolic link from the spool area to the data files rather than trying to copy them (so large files can be printed). This means the data files should not be modified or removed until they have been printed. In the absence of this option, files larger than 1 Megabyte in length are truncated. Note: the **–s** option only works on the local host (files sent to remote printer hosts are copied anyway), and only with named data files — it doesn't work if **lpr** is at the end of a pipeline.

*filter-option*

The following single letter options notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

**–p** Use **pr** to format the files (**lpr**–p is very much like **pr | lpr**).
**–l** Print control characters and suppress page breaks.
**–t** The files contain **troff**(1) (cat phototypesetter) binary data.
**–n** The files contain data from **ditroff** (device independent troff).
**–d** The files contain data from **tex** (DVI format from Stanford).
**–g** The files contain standard plot data as produced by the **plot**(3X) routines (see also **plot**(1G) for the filters used by the printer spooler).
**–v** The files contain a raster image, see **rasterfile** (5).
**–c** The files contain data produced by *cifplot*.
**–f** Interpret the first character of each line as a standard FORTRAN carriage control character.

If no *filter-option* is given (and the printer can interpret PostScript), the string '**%!**' as the first two characters of a file indicates that it contains PostScript commands.

FILES

| | |
|---|---|
| **/etc/passwd** | personal identification |
| **/etc/printcap** | printer capabilities data base |
| **/usr/lib/lpd** | line printer daemon |
| **/var/spool/l*** | directories used for spooling |
| **/var/spool/l*/cf*** | daemon control files |
| **/var/spool/l*/df*** | data files specified in '**cf**' files |
| **/var/spool/l*/tf*** | temporary copies of '**cf**' files |
| **/usr/lib/vfont/font** | |

SEE ALSO

**lpq**(1), **lprm**(1), **plot**(1G), **pr**(1V), **screendump**(1), **troff**(1), **plot**(3X), **printcap**(5), **rasterfile**(5), **lpc**(8), **lpd**(8)

DIAGNOSTICS

**lpr: copy file is too large**

A file is determined to be too "large" to print by copying into the spool area. Use the **–s** option as defined above to make a symbolic link to the file instead of copying it. A too-large file is approximately 1 Megabyte. **lpr** truncates the file, and prints as much of it as it can.

**lpr:** *printer*: **unknown printer**

The **printer** was not found in the **printcap** database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the **/etc/printcap** file.

**lpr:** *printer*: **jobs queued, but cannot start daemon.**

The connection to **lpd** on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket **/dev/printer** to be sure it still exists (if it does not exist, there is no **lpd** process running).

**lpr:** *printer*: **printer queue is disabled**

This means the queue was turned off with

**example% /usr/etc/lpc disable** *printer*

to prevent **lpr** from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with **lpc**.

If a connection to **lpd** on the local machine cannot be made **lpr** will say that the daemon cannot be started. Diagnostics may be printed in the daemon's log file regarding missing spool files by **lpd**

## BUGS

Command-line options cannot be combined into a single argument as with some other commands. The command:

**lpr -fs**

is not equivalent to

**lpr -f -s.**

copies the file to the spooling directory rather than making a symbolic link. Placing the **-s** flag first, or writing each as a separate argument makes a link as expected.

**lpr -p** is not precisely equivalent to **pr|lpr**. **lpr -p** puts the current date at the top of each page, rather than the date last modified.

Fonts for **troff**(1) and *tex* reside on the printer host. It is currently not possible to use local font libraries.

If you spool too large a file, output is truncated at about 1 Megabyte.

**lpr** objects to printing binary files.

The **-s** option only works for print jobs that are run from the printer host itself. Jobs added to the queue from a remote host are always copied into the spool area. That is, if the printer does not reside on the host that **lpr** is run from, the spooling system makes a copy the file to print, and places it in the spool area of the printer host, regardless of **-s**.

NAME

lprm – remove jobs from the printer queue

SYNOPSIS

**lprm** [ –P*printer* ] [ – ] [ *job #* ... ] [ *username* ... ]

DESCRIPTION

**lprm** removes a job or jobs from a printer's spooling queue. Since the spool directory is protected from users, using **lprm** is normally the only method by which a user can remove a job.

Without any arguments, **lprm** deletes the job that is currently active, provided that the user who invoked **lprm** owns that job.

When the super-user specifies a *username*, **lprm** removes all jobs belonging to that user.

You can remove a specific job by supplying its job number as an argument, which you can obtain using **lpq**(1). For example:

```
example% lpq –Phost
host is ready and printing
Rank   Owner  Job  Files                 Total Size
active wendy  385  standard input        35501 bytes
example% lprm –Phost 385
```

**lprm** reports the names of any files it removes, and is silent if there are no applicable jobs to remove.

**lprm** kills the active printer daemon, if necessary, before removing spooled jobs; it restarts the daemon when through.

OPTIONS

–P*printer*

Specify the queue associated with a specific printer. Otherwise the value of the **PRINTER** variable in the environment is used. If this variable is unset, the queue for the default printer is used.

–

Remove all jobs owned by you. If invoked by the super-user, all jobs in the spool are removed. (Job ownership is determined by the user's login name and host name on the machine where the **lpr** command was invoked).

FILES

| | |
|---|---|
| **/etc/printcap** | printer characteristics file |
| **/var/spool/*** | spooling directories |
| **/var/spool/l*/lock** | lock file used to obtain the pid of the current daemon and the job number of the currently active job |

SEE ALSO

**lpr**(1), **lpq**(1), **lpd**(8)

DIAGNOSTICS

**lprm:** *printer* **: cannot restart printer daemon**

The connection to **lpd** on the local machine failed. This usually means the printer server started at boot time has died or is hung. If it is hung, the master **lpd**(8) daemon may have to be killed and a new one started.

BUGS

Since race conditions are possible when updating the lock file, an active job may be incorrectly identified for removal by an **lprm** command issued with no arguments. During the interval between an **lpq**(1) command and the execution of **lprm**, the next job in line may have become active; that job may be removed unintentionally if it is owned by you. To avoid this, supply **lprm** with the job number to remove when a critical job that you own is next in line.

**NAME**

lptest – generate lineprinter ripple pattern

**SYNOPSIS**

**lptest** [ *length* [ *count* ] ]

**DESCRIPTION**

**lptest** writes the traditional "ripple test" pattern on standard output. In 96 lines, this pattern will print all 96 printable ASCII characters in each position. While originally created to test printers, it is quite useful for testing terminals, driving terminal ports for debugging purposes, or any other task where a quick supply of random data is needed.

The *length* argument specifies the output line length if the the default length of 79 is inappropriate.

The *count* argument specifies the number of output lines to be generated if the default count of 200 is inappropriate. Note that if *count* is to be specified, *length* must be also be specified.

NAME

ls – list the contents of a directory

SYNOPSIS

**ls** [ **–aAcCdfFgilLqrRstu1** ] *filename* ...

SYSTEM V SYNOPSIS

**/usr/5bin/ls** [ **–abcCdfFgilLmnopqrRstux** ] *filename* ...

DESCRIPTION

For each *filename* which is a directory, **ls** lists the contents of the directory; for each *filename* which is a file, **ls** repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

In order to determine output formats for the **–C**, **–x**, and **–m** options, **/usr/5bin/ls** uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the **terminfo** database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

**Permissions Field**

The mode printed under the **–l** option contains 10 characters interpreted as follows. If the first character is:

- **d** entry is a directory;
- **b** entry is a block-type special file;
- **c** entry is a character-type special file;
- **l** entry is a symbolic link;
- **p** entry is a FIFO (also known as "named pipe") special file;
- **s** entry is an **AF_UNIX** address family socket, or
- – entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next refers to permissions to others in the same user-group; and the last refers to all others. Within each set the three characters indicate permission respectively to read, towrite, or to execute the file as a program. For a directory, "execute" permission is interpreted to mean permission to search the directory.The permissions are indicated as follows:

- **r** the file is readable;
- **w** the file is writable;
- **x** the file is executable;
- – the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the owner-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally **x** or '–') is **t** if the 1000 bit of the mode is on. See **chmod**(1V) for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

OPTIONS

**–a** List all entries; in the absence of this option, entries whose names begin with a '.' are *not* listed (except for the super-user, for whom **ls**, but not **/usr/5bin/ls**, normally prints even files that begin with a '.').

**–A** (**ls** only) Same as **–a**, except that '.' and '..' are not listed.

**–c** Use time of last edit (or last mode change) for sorting or printing.

**–C** Force multi-column output, with entries sorted down the columns; for **ls**, this is the default when output is to a terminal.

**–d** If argument is a directory, list only its name (not its contents); often used with **–l** to get the status of a directory.

**–f** Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **–l**, **–t**, **–s**, and **–r**, and turns on **–a**; the order is the order in which entries appear in the directory.

**–F** Mark directories with a trailing slash ('/'), executable files with a trailing asterisk ('*'), symbolic links with a trailing at-sign ('@'), and AF_UNIX address family sockets with a trailing equals sign ('=').

**–g** For **ls**, show the group ownership of the file in a long output. For **/usr/5bin/ls**, print a long listing, the same as **–l**, except that the owner is not printed.

**–i** For each file, print the i-number in the first column of the report.

**–l** List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '**—>**'. **/usr/5bin/ls** will print the group in addition to the owner.

**–L** If argument is a symbolic link, list the file or directory the link references rather than the link itself.

**–q** Display non-graphic characters in filenames as the character **?**; this is the default when output is to a terminal.

**–r** Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

**–R** Recursively list subdirectories encountered.

**–s** Give size of each file, including any indirect blocks used to map the file, in kilobytes (**ls**) or 512-byte blocks (**/usr/5bin/ls**).

**–t** Sort by time modified (latest first) instead of by name.

**–u** Use time of last access instead of last modification for sorting (with the **–t** option) and/or printing (with the **–l** option).

**–1** (**ls** only) Force one entry per line output format; this is the default when output is not to a terminal.

SYSTEM V OPTIONS

**–b** Force printing of non-graphic characters to be in the octal **\ddd** notation.

**–m** Stream output format; the file names are printed as a list separated by commas, with as many entries as possible printed on a line.

**–n** The same as **–l**, except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

**–o** The same as **–l**, except that the group is not printed.

**–p** Put a slash ('/') after each filename if that file is a directory.

**–x** Multi-column output with entries sorted across rather than down the page.

**FILES**

| | |
|---|---|
| **/etc/passwd** | to get user ID's for '**ls –l**' and '**ls –o**'. |
| **/etc/group** | to get group ID for '**ls –g**' and '**/usr/5bin/ls –l**'. |
| **/usr/share/lib/terminfo/*** | to get terminal information for **/usr/5bin/ls.** |

**BUGS**

NEWLINE and TAB are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as '**ls –s**' is much different than '**ls –s | lpr**'. On the other hand, not doing this setting would make old shell scripts which used **ls** almost certain losers.

None of the above apply to **/usr/5bin/ls.**

Unprintable characters in file names may confuse the columnar output options.

## NAME

m4 – macro language processor

## SYNOPSIS

**m4** [ *filename* ] ...

## SYSTEM V SYNOPSIS

**/usr/5bin/m4** [ **–es** ] [ **–B***int* ] [ **–H***int* ] [ **–S***int* ] [ **–T***int* ] [ **–D***name=val* ] [ **–U***name* ] [ *filename* ]...

## DESCRIPTION

**m4** is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is '–', the standard input is read. The processed text is written on the standard output.

Macro calls have the form:

*name*(*argument1*[, *argument2*, ...,] *argumentn*)

The '(' must immediately follow the name of the macro. If the name of a defined macro is not followed by a '(', it is interpreted as a call of the macro with no arguments. Potential macro names consist of letters, digits, and '_', (underscores) where the first character is not a digit.

Leading unquoted SPACE, TAB, and NEWLINE characters are ignored while collecting arguments. Left and right single quotes (“”) are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, the arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be NULL. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

## SYSTEM V OPTIONS

The options and their effects are as follows:

**–e** Operate interactively. Interrupts are ignored and the output is unbuffered.

**–s** Enable line sync output for the C preprocessor (#line ...)

**–B***int* Change the size of the push-back and argument collection buffers from the default of 4,096.

**–H***int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

**–S***int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

**–T***int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **–D** or **–U** flags:

**–D***name*[=*val*]
Define *filename* to be **val** or to be NULL in *val*'s absence.

**–U***name*
Undefine *name*.

## USAGE

### Built-In Macros

**m4** makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are NULL unless otherwise stated.

**define** The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*'th argument. Argument 0 is the name of the macro; missing arguments are replaced by the NULL string.

**undefine** Remove the definition of the macro named in the argument.

**ifdef** If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is NULL. The word *unix* is predefined.

**changequote** Change quote characters to the first and second arguments. **changequote** without arguments restores the original values (that is, "").

**divert** **m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to the (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

**undivert** Display immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

**divnum** Return the value of the current output stream.

**dnl** Read and discard characters up to and including the next NEWLINE.

**ifelse** Has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6, 7 and so on. Otherwise, the value is either the last string not used by the above process, or, if it is not present, NULL.

**incr** Return the value of the argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

**eval** Evaluate the argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation); relationals; parentheses.

**len** Return the number of characters in the argument.

**index** Return the position in the first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

**substr** Return a substring of the first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

**translit** Transliterate the characters in the first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

**include** Return the contents of the file named in the argument.

**sinclude** Is similar to **include**, except that it says nothing if the file is inaccessible.

**syscmd** Execute the system command given in the first argument. No value is returned.

**maketemp** Fill in a string of XXXXX in the argument with the current process ID.

**errprint** Print the argument on the diagnostic output file.

**dumpdef** Print current names and definitions, for the named items, or for all if no arguments are given.

## SYSTEM V USAGE

In the System V version of **m4**, the following built-in macros have added capabilities.

### Built-In Macros

**define** '$#' is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like '$*', but each argument is quoted (with the current quotes).

**changequote** Change quote symbols to the first and second arguments. The symbols may be up to five characters long.

**eval** Additional operators include bitwise '&', '|', '^' and ' '. Octal, decimal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

The System V version of **m4** makes available the following additional built-in macros.

**defn** Return the quoted definition of the argument(s). It is useful for renaming macros, especially built-ins.

**pushdef** Like **define**, but saves any previous definition.

**popdef** Remove current definition of the argument(s), exposing the previous one, if any.

**shift** Return all but the first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

**changecom** Change left and right comment markers from the default # and NEWLINE. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes NEWLINE. With two arguments, both markers are affected. Comment markers may be up to five characters long.

**decr** Return the value of the argument decremented by 1.

**sysval** Return code from the last call to **syscmd**.

**m4exit** Exit immediately from **m4**. Argument 1, if given, is the exit code; the default is 0.

**m4wrap** Argument 1 will be pushed back at final EOF. For example, '**m4wrap("cleanup()")**'.

**traceon** With no arguments, turn on tracing for all macros (including built-ins). Otherwise, turn on tracing for named macros.

**traceoff** Turn off trace globally and for any macros specified. Macros specifically traced by **traceon** can be untraced only by specific calls to **traceoff**.

SEE ALSO

*m4 — A Macro Processor*, in *Programming Utilities and Libraries*

## NAME

mach – display the processor type of the current host

## SYNOPSIS

**mach**

## DESCRIPTION

The **mach** command displays the processor-type of the current Sun host.

## SEE ALSO

**arch**(1), **machid**(1)

NAME

machid, sun, iAPX286, m68k, pdp11, sparc, u3b, u3b2, u3b5, u3b15, vax, i386 – return a true exit status if the processor is of the indicated type

SYNOPSIS

**sun**

**iAPX286**

**m68k**

**pdp11**

**sparc**

**u3b**

**u3b2**

**u3b5**

**u3b15**

**vax**

**i386**

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

| | |
|---|---|
| **sun** | True if you are on a Sun system. |
| **iAPX286** | True if you are on a computer using an iAPX286 processor. |
| **i386** | True if you are on a computer using an iAPX386 processor. |
| **m68k** | True if you are on a computer, such as a Sun-2 or a Sun-3, using an M68000-family processor. |
| **pdp11** | True if you are on a PDP-11. |
| **sparc** | True if you are on a computer, such as a Sun-4, using a SPARC-family processor. |
| **u3b** | True if you are on a 3B20S computer. |
| **u3b2** | True if you are on a 3B2 computer. |
| **u3b5** | True if you are on a 3B5 computer. |
| **u3b15** | True if you are on a 3B15 computer. |
| vax | True if you are on a VAX. |

The commands that do not apply will return a false (non-zero) value. These commands are often used within **make**(1) makefiles and shell procedures to increase portability.

SEE ALSO

**arch**(1), **mach**(1), **make**(1), **sh**(1), **test**(1V), **true**(1)

## NAME

mail, Mail – read or send mail messages

## SYNOPSIS

**Mail** [ **–deHinNUv** ] [ **–f** [ *filename* |**+***folder* ] ] [ **–T** *file* ] [ **–u** *user* ]

**Mail** [ **–dFinUv** ] [ **–h** *number* ] [ **–r** *address* ] [ **–s** *subject* ] *recipient* ...

**/usr/ucb/mail** ...

## DESCRIPTION

**mail** is a comfortable, flexible, interactive program for composing, sending and receiving electronic messages. While reading messages, **mail** provides you with commands to browse, display, save, delete, and respond to messages. While sending mail, **mail** allows editing and reviewing of messages being composed, and the inclusion of text from files or other messages.

Incoming mail is stored in the *system mailbox* for each user. This is a file named after the user in **/var/spool/mail**. **mail** normally looks in this file for incoming messages, but you can use the **MAIL** environment variable to have it look in a different file. When you read a message, it is marked to be moved to a secondary file for storage. This secondary file, called the *mbox*, is normally the file **.mbox** in your home directory. This file can also be changed by setting the **MBOX** environment variable. Messages remain in the *mbox* file until deliberately removed.

## OPTIONS

If no *recipient* is specified, **mail** attempts to read messages from the system mailbox.

| | |
|---|---|
| **–d** | Turn on debugging output. (Neither particularly interesting nor recommended.) |
| **–e** | Test for presence of mail. If there is no mail, **mail** prints nothing and exits (with a successful return code). |
| **–F** | Record the message in a file named after the first recipient. Override the **record** variable, if set. |
| **–H** | Print header summary only. |
| **–i** | Ignore interrupts (as with the **ignore** variable). |
| **–n** | Do not initialize from the system default **Mail.rc** file. |
| **–N** | Do not print initial header summary. |
| **–U** | Convert **uucp** style addresses to Internet standards. Overrides the **conv** environment variable. |
| **–v** | Pass the **–v** flag to **sendmail**(8). |
| **–f** [*filename*] | Read messages from *filename* instead of system mailbox. If no *filename* is specified, the *mbox* is used. |
| **–f** [ *+folder*] | Use the file *folder* in the folder directory (same as the **folder** command). The name of this directory is listed in the **folder** variable. |
| **–h** *number* | The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. |
| **–r** *address* | Pass *address* to network delivery software. All tilde (~) commands are disabled. |
| **–s** *subject* | Set the **Subject** header field to *subject*. |
| **–T** *file* | Print the contents of the *article-id* fields of all messages that were read or deleted on *file* (for the use of network news programs if available). |
| **–u** *user* | Read *user*'s system mailbox. This is only effective if *user*'s system mailbox is not read protected. |

## USAGE

Refer to *Mail and Messages: Beginner's Guide* for tutorial information about **mail**.

### Starting Mail

As it starts, **mail** reads commands from a system-wide file (**/usr/lib/Mail.rc**) to initialize certain variables, then it reads from from a private start-up file called the *.mailrc* file (it is normally the file **.mailrc** in your home directory, but can be changed by setting the MAILRC environment variable) for your personal commands and variable settings. Most **mail** commands are legal inside start-up files. The most common uses for this file are to set up initial display options and alias lists. The following commands are *not* legal in the start-up file: **!**, **Copy**, **edit**, **followup**, **Followup**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **replyall**, **replysender**, **shell**, and **visual**. Any errors in the start-up file cause the remaining lines in that file to be ignored.

You can use the **mail** command to send a message directly by including names of recipients as arguments on the command line. When no recipients appear on the **mail** command line, it enters command mode, from which you can read messages sent to you. If you list no recipients and have no messages, **mail** prints the message: '**No mail for** *username*' and exits.

When in command mode (while reading messages), you can send messages using the **mail** command.

### Sending Mail

While you are composing a message to send, **mail** is in *input* mode. If no subject is specified as an argument to the command a prompt for the subject is printed. After entering the subject line, **mail** enters *input* mode to accept the text of your message to send.

As you type in the message, **mail** stores it in a temporary file. To review or modify the message, enter the appropriate *tilde escapes*, listed below, at the beginning of an input line.

To indicate that the message is ready to send, type a dot (or EOF character, normally CTRL-D ) on a line by itself. **mail** submits the message to **sendmail**(8) for routing to each *recipient*.

Recipients can be local usernames, Internet addresses of the form:

*name*@*domain*

**uucp**(1C) addresses of the form:

... [*host*!]*host*!*username*

filenames for which you have write permission, or alias groups. If the name of the *recipient* begins with a pipe symbol (|), the remainder of the name is taken as a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as **lpr**(1) to record outgoing mail on paper. An alias group is the name of a list of recipients that is set by the **alias** command, taken from the host's **/etc/aliases** file, or taken from the Yellow Pages aliases domain. See **aliases**(5) for more information about mail addresses and aliases.

### Tilde Escapes

The following *tilde escape* commands can be used when composing messages to send. Each must appear at the beginning of an input line. The escape character (~), can be changed by setting a new value for the **escape** variable. The escape character can be entered as text by typing it twice.

~! [*shell-command*]
: Escape to the shell. If present, run *shell-command*.

~. : Simulate EOF (terminate message input).

~: *mail-command*
~_ *mail-command*
: Perform the indicated **mail** command. Valid only when sending a message while reading mail.

~? : Print a summary of tilde escapes.

~**A** : Insert the autograph string **Sign** into the message.

~**a** : Insert the autograph string **sign** into the message.

**~b** *name* ...
Add the *name*s to the blind carbon copy (**Bcc**) list. This is like the carbon copy (**Cc**) list, except that the names in the **Bcc** list are not shown in the header of the mail message.

**~c** *name* ...
Add the *name*s to the carbon copy (**Cc**) list.

**~d** Read in the *dead.letter* file. The name of this file is listed in the variable **DEAD**.

**~e** Invoke the editor to edit the message. The name of the editor is listed in the **EDITOR** variable. The default editor is **ex**(1).

**~f** [*message-list*]
Forward the listed messages, or the current message being read. Valid only when sending a message while reading mail; the messages are inserted without alteration (as opposed to the **~m** escape).

**~h** Prompt for the message header lines: **Subject**, **To**, **Cc**, and **Bcc**. If the header line contains text, you can edit the text by backspacing over it and retyping.

**~i** *variable*
Insert the value of the named *variable* into the message.

**~m** [*message-list*]
Insert text from the specified messages, or the current message, into the letter. Valid only when sending a message while reading mail; the text the message is shifted to the right, and the string contained in the **indentprefix** variable is inserted as the leftmost characters of each line. If **indentprefix** is not set, a TAB character is inserted into each line.

**~p** Print the message being entered.

**~q** Quit from input mode by simulating an interrupt. If the body of the message is not empty, the partial message is saved in the *dead.letter* file.

**~r** *filename*
**~<** *filename*
**~<!** *shell-command*
Read in text from the specified file or the standard output of the specified *shell-command*.

**~s** *subject*
Set the subject line to *subject*.

**~t** *name* ...
Add each *name* to the list of recipients.

**~v** Invoke a visual editor to edit the message. The name of the editor is listed in the **VISUAL** variable. The default visual editor is **vi**(1).

**~w** *filename*
Write the message text onto the given file, without the header.

**~x** Exit as with **~q** but do not save the message in the *dead.letter* file.

**~|** *shell-command*
Pipe the body of the message through the given *shell-command*. If *shell-command* returns a successful exit status, the output of the command replaces the message.

**Reading Mail**

When you enter *command* mode in order to read your messages, **mail** displays a header summary of the first several messages, followed by a prompt for one of the commands listed below. The default prompt is the **&** (ampersand character).

Message are listed and referred to by number. There is, at any time, a **current** message, which is marked by a > in the header summary. For commands that take an optional list of messages, if you omit a message number as an argument, the command applies to the current message.

A *message-list* is a list of message specifications, separated by SPACE characters, which may include:

| | |
|---|---|
| . | The current message. |
| *n* | Message number *n*. |
| ^ | The first undeleted message. |
| $ | The last message. |
| + | The next undeleted message. |
| – | The previous undeleted message. |
| * | All messages. |
| *n–m* | An inclusive range of message numbers. |
| *user* | All messages from *user*. |
| */string* | All messages with *string* in the subject line (case ignored). |
| :*c* | All messages of type *c*, where *c* is one of: |

| | |
|---|---|
| **d** | deleted messages |
| **n** | new messages |
| **o** | old messages |
| **r** | read messages |
| **u** | unread messages |

Note: the context of the command determines whether this type of message specification makes sense.

Additional arguments are treated as strings whose usage depends on the command involved. Filenames, where expected, are expanded using the normal shell filename-substitution mechanism.

Special characters, recognized by certain commands, are documented with those commands.

**Commands**

While in command mode, if you type in an empty command line (a RETURN or NEWLINE only), the **print** command is assumed. The following is a complete list of **mail** commands:

! *shell-command* — Escape to the shell. The name of the shell to use is listed in the **SHELL** variable.

*# arguments* — Null command. This may be used as if it were a comment in *.mailrc* files, but note that it must be separated from its arguments (commentary) by white space.

= — Print the current message number.

? — Print a summary of commands.

**alias** [*alias recipient* ...]
**group** [*alias recipient* ...]
Declare an alias for the given list of recipients. The list will be substituted when the *alias* is used as a recipient while sending mail. When put in the *.mailrc* file, this command provides you with a record of the alias. With no arguments, the command displays the list of defined aliases.

**alternates** *name* ... Declare a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alternates** prints the current list of alternate names.

**cd**[ *directory*]
**chdir** [*directory*] Change directory. If *directory* is not specified, $HOME is used.

**copy** [*message-list*] [*filename*]
Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.

Copy [*message-list*] Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the **S**ave command.

**d**elete [*message-list*] Delete messages from the system mailbox. If the variable **autoprint** is set, print the message following the last message deleted.

**d**iscard [*header-field*...]
**i**gnore [*header-field*...]
Suppress printing of the specified header fields when displaying messages on the screen, such as "Status" and "Received". The fields are included when the message is saved unless the variable **alwaysignore** is set. The **P**rint and **T**ype commands display all header fields, ignored or not.

**dp** [*message-list*]
**dt** [*message-list*]
Delete the specified messages from the system mailbox, and print the message after the last one deleted. Equivalent to a **d**elete command followed by a **p**rint command.

**e**cho [*string* ...] Echo the given strings (like **echo**(1V)).

**e**dit [*message-list*] Edit the given messages. The messages are placed in a temporary file and the **EDITOR** variable is used to get the name of the editor. The default editor is **ex**(1).

**exit**
**x**it Exit from **mail** without changing the system mailbox. No messages are saved in the *mbox* (see also **q**uit).

**file** [*filename*]
**fo**lder [*filename*] Quit from the current mailbox file and read in the named mailbox file. Several special characters are recognized when used as file names:

| | |
|---|---|
| *%* | Your system mailbox. |
| *%user* | The system mailbox for *user*. |
| *#* | The previous mail file. |
| *&* | Your *mbox* file (of messages previously read). |
| *+filename* | The named file in the *folder* directory (listed in the **folder** variable). |

With no arguments, **file** prints the name of the current mail file, and the number of messages and characters it contains.

**folders** Print the name of each mail file in the *folder* directory (listed in the **folder** variable).

**f**ollowup [*message*] Respond to a message, recording the response in a file, name of which is derived from the author of the message (overrides the **record** variable, if set). See also the **F**ollowup, **S**ave, and **C**opy commands and the **outfolder** variable.

**F**ollowup [*message-list*]
Respond to the first message in the message list, sending the message to the author of each message in the list. The subject line is taken from the first message, and the response is recorded in a file, the name of which is derived from the author of the first message (overrides the **record** variable, if set). See also the **f**ollowup, **S**ave, and **C**opy commands and the **outfolder** variable.

**f**rom [*message-list*] Print the header summary for the indicated messages or the current message.

**g**roup *alias name* ... Same as the **a**lias command.

**h**eaders [*message*] Print the page of headers that includes the message specified, or the current message. The **screen** variable sets the number of headers per page. See also the **z** command.

**help** Print a summary of commands.

**hold** [*message-list*]
**preserve** [*message-list*]
Hold the specified messages in the system mailbox.

**if** *s* | *r* | *t*
*mail-command*
...
**else**
*mail-command*
...
**endif** Conditional execution, where *s* will execute following *mail-command* up to an **else** or **endif**, if the program is in *send* mode, *r* executes the *mail-command* only in *receive* mode, and *t* executes the *mail-command* only if **mail** is being run from a terminal. Useful primarily in the *.mailrc* file.

**ignore** [*header-field*...]
Same as the **discard** command.

**inc** Incorporate messages that arrive while you are reading the system mailbox. The new messages are added to the message list in the current **mail** session. This command does not commit changes made during the session, and prior messages are not renumbered.

**list** Prints all commands available. No explanation is given.

**load** [*message*] *filename*
Load the specified message from the name file. *filename* should contain a single mail message including mail headers (as saved by the **save** command).

**mail** *recipient* ... Mail a message to the specified recipients.

**mbox** [*message-list*] Arrange for the given messages to end up in the standard *mbox* file when **mail** terminates normally. See also the **exit** and **quit** commands.

**new** [*message-list*]
New [*message-list*]
**unread** [*message-list*]
**Unread** [*message-list*]
Take a message list and mark each message as *not* having been read.

**next** *message* Go to next message matching *message*. A *message-list* can be given instead of *message*, but only first valid message in the list is used. (This can be used, for instance, to jump to the next message from a specific user.)

**pipe** [*message-list*] [*shell-command*]
| [*message-list*] [*shell-command*]
Pipe the message through *shell-command*. The message is treated marked as read (and normally saved to the *mbox* file when **mail** exits). If no arguments are given, the current message is piped through the command specified by the value of the **cmd** variable. If the **page** variable is set, a form feed character is inserted after each message.

**preserve** [*message-list*]
Same as the **hold** command.

**print** [*message-list*]
**type** [*message-listfP*] Print the specified messages. If the **crt** variable is set, messages longer than the number of lines it indicates paged through the command specified by the **PAGER** variable. The default paging command is **more**(1).

**Print** [*message-list*]
**Type** [*message-list*] Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** and **retain** commands.

**quit** Exit from **mail** storing messages that were read in the *mbox* file and unread messages in the system mailbox. Messages that have been explicitly saved in a file are deleted unless the variable **keepsave** is set.

**reply** [*message-list*]
**respond** [*message-list*]
**replysender** [*message-list*]
Send a response to the author of each message in the *message-list*. The subject line is taken from the first message. If **record** is set to a filename, a copy of the the reply is added to that file. If the **replyall** variable is set, the actions of **Reply**/**Respond** and **reply**/**respond** are reversed. The **replysender** command is not affected by the **replyall** variable, but sends each reply only to the sender of each message.

**Reply** [*message*]
**Respond** [*message*]
**replyall** [*message*] Reply to the specified message, including all other recipients of that message. If the variable **record** is set to a filename, a copy of the reply added to that file. If the **replyall** variable is set, the actions of **Reply**/**Respond** and **reply**/**respond** are reversed. The **replyall** command is not affected by the **replyall** variable, but always sends the reply to all recipients of the message.

**retain** Add the list of header fields named to the *retained list*. Only the header fields in the retain list are shown on your terminal when you print a message. All other header fields are suppressed. The set of retained fields specified by the **retain** command overrides any list of ignored fields specified by the **ignore** command. The **Type** and **Print** commands can be used to print a message in its entirety. If **retain** is executed with no arguments, it lists the current set of retained fields.

**save** [*message-list*] [*filename*]
Save the specified messages in the named file. The file is created if it does not exist. If no *filename* is specified, the file named in the **MBOX** variable is used, **mbox** in your home directory by default. Each saved message is deleted from the system mailbox when **mail** terminates unless the **keepsave** variable is set. See also the **exit** and **quit** commands.

**Save** [*message-list*] Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken from the author's name, with all network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and the **outfolder** variables.

**set** [*variable*[=*value*]]
Define a *variable*. To assign a *value* to *variable*, separate the variable name from the value by an '=' (there must be no space before or after the '='). A variable may be given a null, string, or numeric *value*. To embed SPACE characters within a *value* enclose it in quotes.

With no arguments, **set** displays all defined variables and any values they might have. See **Variables** for a description of all predefined **mail** variables.

**shell** Invoke the interactive shell listed in the **SHELL** variable.

**size** [*message-list*] Print the size in characters of the specified messages.

**source** *filename* Read commands from the given file and return to command mode.

**top** [*message-list*] Print the top few lines of the specified messages. If the **toplines** variable is set, it is taken as the number of lines to print. The default number is 5.

**touch** [*message-list*] Touch the specified messages. If any message in *message-list* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See also the **exit** and **quit** commands.

type [*message-list*] Same as the **print** command.

Type [*message-list*] Same as the **Print** command.

**undelete** [*message-list*]
Restore deleted messages. This command only restores messages *deleted in the current mail session.* If the **autoprint** variable is set, the last message restored is printed.

**unread** [*message-list*]
Unread [*message-list*]
Same as the **new** command.

**unset** *variable* ... Erase the specified variables. If the variable was imported from the environment (that is, an environment variable or exported shell variable), it cannot be unset from within **mail**.

version Print the current version and release date of the **mail** utility.

visual [*message-list*] Edit the given messages with the screen editor listed in the **VISUAL** variable. The default screen editor is **vi**(1). Each message is placed in a temporary file for editing.

write [*message-list*] [*filename*]
Write the given messages onto the specified file, but without the header and trailing blank line. Otherwise, this is equivalent to the **save** command.

xit Same as the **exit** command.

z[+|−] Scroll the header display forward (+) or backward (−) one screenful. The number of headers displayed is set by the **screen** variable.

**Forwarding Messages**

To forward a specific message, include it in a message to the desired recipients with the **~f** or **~m** tilde escapes. To forward mail automatically, add a comma-separated list of addresses for additional recipients to the **.forward** file in your home directory. (This is different from the format of the **alias** command, which takes a space-separated list instead.) Note: forwarding addresses must be valid (as described in **aliases**(5)), or the messages will "bounce." You cannot, for instance, reroute your mail to a new host by forwarding it to your new address if it is not yet listed in the YP aliases domain.

**Variables**

The behavior of **mail** is governed by a set of predefined variables that are set and cleared using the **set** and **unset** commands.

*Environment Variables*

Values for the following variables are read in automatically from the environment; they cannot be altered from within **mail**:

**HOME**=*directory*
The user's home directory.

**MAIL**=*filename*
The name of the initial mailbox file to read (in lieu of the standard system mailbox). The default is **/var/spool/mail/***username* .

**MAILRC**=*filename*
The name of the personal start-up file. The default is **$HOME/.mailrc**.

*Mail Variables*

The following variables can be initialized within the *.mailrc* file, or set and altered interactively using the **set** command. They can also be imported from the environment (in which case their values cannot be changed within **mail**). The **unset** command clears variables. The set command can also be used to clear a variable by prefixing the word **no** to the name of the variable to clear.

Variables for which values are normally supplied are indicated with an equal-sign (=). The equal-sign is required by the **set** command, and there can be no spaces between the variable-name, equal-sign, and value, using **set** to assign a value.

**allnet** All network names whose last component (login name) match are treated as identical. This causes the message list specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the **metoo** variable.

**alwaysignore** Ignore header fields with **ignore** everywhere, not just during **print** or **type**. Affects the **save**, **Save**, **copy**, **Copy**, **top**, **pipe**, and **write** commands, and the **~m** and **~f** tilde escapes.

**append** Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend** but **append** is set in the global start-up file (which can be suppressed with the **-n** command line option).

**askcc** Prompt for the **Cc** list after message is entered. Default is **noaskcc**.

**asksub** Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

**autoprint** Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

**bang** Enable the special-casing of exclamation points (!) in shell escape command lines as in **vi**(1). Default is **nobang**.

**cmd**=*shell-command*
Set the default command for the **pipe** command. No default value.

**conv**=*conversion*
Convert **uucp** addresses to the address style specified by *conversion*, which can be either:

**internet**
This requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing.

**optimize**
Remove loops in **uucp**(1C) address paths (typically generated by the **reply** command). No rerouting is performed; **mail** has no knowledge of UUCP routes or connections.

Conversion is disabled by default. See also **sendmail**(8) and the **-U** command line option.

**crt**=*number* Pipe messages having more than *number* lines through the command specified by the value of the **PAGER** variable (*more* by default). Disabled by default.

**DEAD**=*filename*
The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is the file **dead.letter** in your home directory.

**debug** Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

**dot** Take a period on a line by itself during input from a terminal as EOF. Default is **nodot** but **dot** is set in the global start-up file (which can be suppressed with the **-n** command line option).

**EDITOR**=*shell-command*
The command to run when the **edit** or **~e** command is used. Default is **ex(1)**.

**escape**=*c* Substitute *c* for the ~ escape character.

**folder**=*directory*
The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), the value of **HOME** is prepended to it. There is no default for the **folder** variable. See also **outfolder** below.

**header** Enable printing of the header summary when entering **mail**. Enabled by default.

**hold** Preserve all messages that are read in the system mailbox instead of putting them in the standard *mbox* save file. Default is **nohold**.

**ignore** Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

**ignoreeof** Ignore EOF during message input. Input must be terminated by a period ('.') on a line by itself or by the '~'. command. Default is **noignoreeof**. See also **dot** above.

**indentprefix**=*string*
When **indentprefix** is set, *string* is used to mark indented lines from messages included with **~m**. The default is a TAB character.

**keep** When the system mailbox is empty, truncate it to zero length instead of removing it. Disabled by default.

**keepsave** Keep messages that have been saved in other files in the system mailbox instead of deleting them. Default is **nokeepsave**.

**LISTER**=*shell-command*
The command (and options) to use when listing the files in the **folder** directory. The default is **ls**(1V).

**MBOX**=*filename*
The name of the file to save messages which have been read. The **xit** command overrides this variable, as does saving the message explicitly to another file. Default is the file **mbox** in your home directory.

**metoo** If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

**no** When used as a prefix to a variable name, has the effect of unsetting the variable.

**onehop** When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one "hop" away).

**outfolder** Locate the files used to record outgoing messages in the directory specified by the **folder** variable unless the pathname is absolute. Default is **nooutfolder**. See **folder** above and the **Save**, **Copy**, **followup**, and **Followup** commands.

**page** Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

**PAGER**=*shell-command*
The command to use as a filter for paginating output, along with any options to be used. Default is **more**(1).

**prompt**=*string* Set the *command mode* prompt to *string*. Default is '**&**'.

**quiet** Refrain from printing the opening message and version when entering **mail**. Default is **noquiet**.

**record=***filename*
Record all outgoing mail in *filename*. Disabled by default. See also the variable **outfolder**.

**replyall** Reverse the effect of the **reply** and **Reply** commands.

**save** Enable saving of messages in the *dead.letter* file on interrupt or delivery error. See **DEAD** for a description of this file. Enabled by default.

**screen=***number* Set the number of lines in a screen–full of headers for the **headers** command.

**sendmail=***shell-command*
Alternate command for delivering messages. Note: in addition to the expected list of recipients, **mail** also passes the **–i** and **–m**, flags to the command. Since these flags are not appropriate to other commands, you may have to use a shell script that strips them from the arguments list before invoking the desired command.

**sendwait** Wait for background mailer to finish before returning. Default is **nosendwait**.

**SHELL=***shell-command*
The name of a preferred command interpreter. Typically inherited from the environment, the shell is normally the one you always use. Otherwise defaults to **sh**(1).

**showto** When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

**sign=***autograph* The *autograph* text inserted into the message when the **~a** (autograph) command is given. No default (see also the **~i** tilde escape).

**Sign=***autograph* The *autograph* text inserted into the message when the **~A** command is given. No default (see also the **~i** tilde escape).

**toplines=***number*
The number of lines of header to print with the **top** command. Default is 5.

**verbose** Invoke **sendmail** with the **–v** flag.

**VISUAL=***shell-command*
The name of a preferred screen editor. Default is **vi**.

FILES

| | |
|---|---|
| **$HOME/***.mailrc* | personal start-up file |
| **$HOME/***.forward* | list of recipients for automatic forwarding of messages |
| **$HOME/***mbox* | secondary storage file |
| **$HOME/***dead.letter* | undeliverable messages file |
| **/var/spool/mail** | directory for system mailboxes |
| **/usr/lib/Mail.help*** | help message files |
| **/usr/lib/Mail.rc** | global start-up file |
| **/tmp/R[emqsx]*** | temporary files |

SEE ALSO

**biff**(1), **binmail**(1), **echo**(1V), **ex**(1), **fmt**(1), **ls**(1V), **mailtool**(1), **more**(1), **sh**(1), **uucp**(1C), **vacation**(1), **vi**(1), **aliases**(5), **newaliases**(8), **sendmail**(8)

*Mail and Messages: Beginner's Guide*

**mail** is found in **/usr/ucb/Mail**, as a link to **/usr/ucb/mail**. If you wish to use the original (version 6) UNIX mail program, you can find it in **/usr/bin/mail**. Its man page is named **binmail**(1).

## BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

Replies do not always generate correct return addresses. Try resending the errant reply with **onehop** set.

**mail** does not lock your record file. So, if you use a record file and send two or more messages simultaneously, lines from the messages may be interleaved in the record file.

The format for the **alias** command is a space-separated list of recipients, while the format for an alias in either the **.forward** or **/etc/aliases** is a comma-separated list.

## NAME

mailtool – SunView interface for the mail program

## SYNOPSIS

**mailtool** [ **–Mx** ] [ **–Mi** *interval* ] [ *generic-tool-arguments* ]

## AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**mailtool** is the standard SunView interface to **mail**(1). It provides a window and mouse-based interface for reading, storing, composing, and sending mail messages. Scrollable windows allow easy access to your mailbox and mail folders. Software "panel buttons" make frequently-used commands readily available. Less-used commands are accessible from menus, and keyboard accelerators are provided for the more experienced user.

The full editing capabilities of **textedit**(1) and the SunView selection service are available for modifying and composing mail. In addition, you can customize **mailtool** by setting various parameters with **defaultsedit**(1).

## OPTIONS

**–Mx** Expert mode. Do not ask for confirmation after potentially damaging **mail** commands. This has the same effect as setting the **expert** variable.

**–Mi** *interval* Check for new mail every *interval* seconds. This has the same effect as specifying a value for the *interval* variable.

*generic-tool-arguments*
**mailtool** accepts the generic tool arguments described in **sunview**(1).

## USAGE

Users who are not familiar with the **mail** command should read *Mail and Messages: Beginner's Guide*. For more information on text editing and the selection service, see the *SunView 1 Beginner's Guide*.

**mailtool** comes up closed. You can open the tool by clicking on the icon with the LEFT mouse button. **mailtool** starts reading your system mailbox as it opens. Alternatively, the frame menu on the icon contains the **Open** pull-right item, which allows you to open **mailtool** to a selected folder, or open and **Compose**, or just open without performing any other operation.

### Subwindows

**mailtool** is composed of six parts. From top to bottom, they are:

frame header
This is the broad stripe at the top of the tool, and it displays status information. The right side displays information about the most-recently executed command. The left side displays other information such as the name of the current folder. When involved in a lengthy operation, **mailtool** displays a message to that effect on the left side. While an operation is pending, the cursor takes on the shape of an hourglass; you must wait until it finishes.

header-list window
This read-only text window contains a list of message headers from the current folder or mailbox. Initially, it shows the contents of your system mailbox (by default). Each header typically contains fields indicating who the message is from, its subject, and so forth. There is a scrollbar to the left that you can use to scroll through the headers.

control panel
This panel contains a collection of software buttons corresponding to the most frequently used **mail**(1) commands. Clicking on one of these buttons starts the indicated operation for either the selected or the current message. Commands that require the name of a folder, such as **Save**, use the contents of the '**File:**' text item. You must enter the name of a file or

folder in this space before clicking on the button.

In addition to the panel buttons, other commands and variations are accessible through menus "behind" each panel button. To display a button menu, hold down the RIGHT mouse button over the panel button. See **Command Menus**, below, for details.

The message window.

This text window displays the current message (marked with '>' in the header window). You can edit this message, in which case the result replaces the original message in the mailbox or folder.

composition panel

This panel contains software buttons for composing or replying to messages, and is visible only when you are composing a message or reply.

composition window

This text window, in which you compose messages to send, appears in conjunction with the composition panel. It is normally displayed only when composing a message or reply. It is initially loaded with mail header lines such as **'To:'** and **'Subject:'**, and perhaps, **'Cc:'**. After these labels come various text fields, such as **|>recipients<|**. You can move from field to field using META-TAB. This advances to the next text field. If you supply input to the field, your input replaces its contents. Empty fields are deleted when the message is sent. You can continue to edit the message as you see fit until you click on **Deliver**, at which point the message is sent as is. (You normally cannot retrieve a message once it has been sent).

pop-up composition window

While composing a message or reply, clicking again on **Compose** or **Reply**, opens another frame that contains a composition panel and window. The only limit on the number of such pop-up composition frames is the number of windows that a tool can support. These additional frames operate independently.

**Basic Mailtool Concepts**

Choosing a message

To choose a message, place the cursor anywhere in its header in the header window and click the LEFT mouse button. If there is no message chosen, **mailtool** applies operations to the current message.

Current message

The message that is displayed in the message window, and flagged with a '>' in the header window.

Confirmation

Some operations require confirmation, in which case an alert is displayed. You can then confirm or cancel the operation.

Folders

When mailtool starts up, it normally reads your system mailbox. However, you can select another "folder" (file containing mail messages) from which to read.

Committing Changes

Some operations change the state of your system mailbox or the current folder. These changes are not finalized until you commit them. For instance, you can "undelete" messages that were deleted, provided that you have not yet done an operation that commits your changes. If the **mailtool** session is interrupted, pending changes to the mailbox or folder do not take effect. The **Done** button commits changes, as does the **Quit** button, which also exits from **mailtool**. To deliberately exit without committing, use the pop-up menu behind the **Quit** button.

**Control Panel**

Except for the **Next** and **Undelete** buttons, **mailtool** commands operate on the selected message or the current message only. You cannot specify a list or range of messages as with **mail**(1). The control panel buttons and items are (in alphabetical order):

**Compose** Open a composition panel and window to compose a message.

**Delete** Delete the selected or current message.

**Done** Commit changes, close **mailtool**, and read new mail on next **Open**.

**'File:'** This is a text item in which to enter the name of a folder for the **Save**, **Copy**, and **Folder** commands. This name can be a full pathname, a pathname relative to the current directory (the directory **mailtool** was started from), or a filename prefixed with a '+' to refer to a file in the "folder" directory.

**Folder** Commit changes and switch to the file or folder specified in the **'File:'** text item.

**Misc** Display a pop-up panel to change the current directory of **mailtool**. Other miscellaneous operations are available on the menu behind this button.

**New Mail** If you are examining your system mailbox, retrieve new mail *without* committing changes. If your are examining a folder, commit any changes to the folder, switch back to system mailbox, and retrieve any new mail in the process.

**Next** Display the message following the *current* message in the message window.

**Print** Print the corresponding message on a hardcopy printer.

**Reply** Open a composition window to reply to the selected or current message.

**Save** Save the current or selected message in the folder specified in the **'File:'** text field, and delete it from your system mailbox or current folder.

**Show** Display the chosen or current message in the message window.

**The Composition Panel**

This panel contains four buttons and a cycle-item. The cycle-item controls the behavior of the composition window when it becomes inactive — when the user delivers or cancels a message. Items in the cycle are:

**Disappear** Remove the composition window and panel from display. This is the default.

**Stay Up** Clear the window, but leave it displayed.

**Close** Close a pop-up composition frame.

The panel buttons are:

**Cancel** Abort the message being composed.

**Deliver** Send the message being composed to the indicated recipients.

**Include** Insert the corresponding message into the composition window at the caret. This operation can be performed repeatedly, to include various messages.

**Re-address** Insert the appropriate **'To:'**, **'Subject:'** and **'Cc:'** fields at the top of the composition window.

**Command Menus**

All panel buttons have menus behind them. The first item on the menu is the default command; choosing this item is the same as clicking on the panel button.

Some menu items are pull-right to menus of related commands. You can browse the button menus to discover what additional commands are available, and what their accelerators are, if any. The following commands are particularly useful.

**Change Directory**

Display a pop-up panel to change the current directory.

**Commit Changes**
Commit changes. This item is behind the **New Mail** button when viewing a folder.

**Commit Changes and Quit**
Behind the **Done** button. Commit changes and exit **mailtool**(1). This is the same as choosing **Quit** from the frame menu.

**Commit Changes and Retrieve New Mail**
Behind the **New Mail** button. Commit changes and retrieve new mail, switching to the system mail box if in a folder. This is the default when viewing a folder.

**Copy** Behind the **Save** button. Copy the selected message to the file or folder specified in the '**File:**' text item, without deleting it from the mailbox or folder.

**Deliver, Leave Window Intact**
Behind the **Deliver** button. Deliver the message, but do not undisplay, close, or clear the message composition window.

**Include, Indented**
Behind the **Include** button. Include the indicated message, setting it off by indention rather than bracketing it with '**--- Begin Included Message ---**' and '**--- End Included Message ---**' lines.

**Previous** Behind the **Next** button. Display the previous message in the message window.

**Quit without Committing Changes**
Behind the **Done** button. Exit **mailtool** *without* committing changes.

**Show Full Header**
Behind the **Show** button. Display the complete message in the message window, including header lines that are normally ignored.

**Source .mailrc**
Behind the **Misc** button. Read in your **.mailrc** file to acquire new variables and settings. Note: this operation does not "forget" the previous option settings; only changes to boolean variables take effect.

**Undelete** Behind the **Delete** button. Undelete the most recently deleted message(s) — this may be used repeatedly. It is inactive when there are no deleted messages.

There are two special menus for use with the '**File:**' text item. Choosing a name from either of these menus replaces the contents of this item. The menu behind the '**File:**' item holds the most recently used folder names of the current session. It is be initialized by the **filemenu** variable. The menu behind the **Folder** button displays all folders in the "folder" directory, which is specified by the **folder** variable (described in **mail**(1)). Folders can be organized into subdirectories within the folder directory. Files in these subdirectories appear in a hierarchy of pull-right menus.

To switch to a folder, choose it from one of the file menus, or type it in directly, and click on the **Folder** button. To return to your system mailbox, use the **New Mail** button.

### Mailtool Variables

In addition to the variables recognized by **mail**(1), **mailtool** recognizes those listed below. They can be set by using **defaultsedit**(1), or by editing your **.mailrc** file directly. Unless otherwise noted, the default for the following variables is off.

**allowreversescan**
When set, allows you to step through messages in latest-first oldest-first order if you choose. The next message depends on the order of travel.

**alwaysusepopup**
Never split the message window to compose or reply; always use pop-up composition windows.

**askbcc** Prompt for the **'Bcc:'** field when composing or replying.

**autoprint** Display the next message when the current message is deleted or saved.

**bell** The number of times to ring the bell when new mail arrives. The default is 0.

**disablefields** Do not use text fields in the composition window. The default is to use text fields.

**editmessagewindow**
Request confirmation before the first editing operation to a mesage in the message window (as opposed to composing a reply). The default is not to request confirmation of the first edit.

**expert** Set expert mode in which no confirmations are requested.

**filemenu** A list of files from which to initialize the **'File:'** menu. These can be absolute pathnames, pathnames relative to the working directory for **mailtool** (typically your home directory), or filenames prefixed with a '+', which are taken as relative to the directory specified in the **folder** variable (see **mail**(1)).

**filemenusize** Specifies the maximum size of the **'File:'** menu. The default is 10.

**flash** The number of times to flash the window or icon when new mail arrives. The default is 0.

**headerlines** The number of lines in header window. The default is 10.

**interval** The interval in seconds to check for new mail. The default is 300.

**maillines** The number of lines in mail message window. The default is 30.

**moveinputfocus**
Move the input focus into the composition window for **Compose** and **Reply**. This only works for click-to-type.

**pop-uplines** The number of lines in pop-up message composition window. The default is 30.

**msgpercent** The percent of the message window to remain visible during **Compose** or **Reply**. The default is 50.

**printmail** The command to use to print a message. The default is **'lpr −p'**.

**trash** The name of trash bin, which may be accessed just like any other folder. If set, all deleted messages are moved to the trash bin. The trash bin is emptied when you commit changes.

**Conditional Settings**

You can make your **.mailrc** set variables conditionally, depending on whether it is running in the tty environment or the window environment. See *Mail and Messages: Beginner's Guide* for details.

FILES

| | |
|---|---|
| **/var/spool/mail/*** | system mailboxes |
| **/.mailrc** | startup file for **mail** and **mailtool** |

SEE ALSO

**binmail**(1), **defaultsedit**(1), **mail**(1), **sunview**(1), **textedit**(1) **aliases**(5), **newaliases**(8), **sendmail**(8)

*Mail and Messages: Beginner's Guide*
*SunView 1 Beginner's Guide*

BUGS

If **mail**(1) receives an error, then **mailtool** may hang, in which case you must kill it.

New mail status is only approximate, therefore the presence of new mail is not always accurately reflected in the icon image or tool frame header.

Mouse input may be lost while **mailtool** switches to iconic state.

When notifying you of new mail, **mailtool** will not flash the window or icon without beeping (ringing the audible bell). Thus, the number of flashes is limited by the number of beeps you set.

## NAME

make – maintain, update, and regenerate related programs and files

## SYNOPSIS

**make** [ **–f** *makefile* ] ... [ **–d** ] [ **–dd** ] [ **–D** ] [ **–DD** ] [ **–e** ] [ **–i** ] [ **–k** ] [ **–n** ] [ **–p** ] [ **–P** ] [ **–q** ] [ **–r** ] [ **–p** ] [ **–s** ] [ **–S** ] [ **–t** ] [ *target* ... ] [ *macro=value* ... ]

## DESCRIPTION

**make** executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries that describe how to bring a target up to date with respect to others on which it depends. These prerequisite targets are called *dependencies*. Since each dependency is a target, it may have dependencies of its own.

Targets, dependencies, and sub-dependencies comprise a tree structure that **make** traces when deciding whether or not to rebuild a *target*. **make** recursively checks each *target* against its dependencies, beginning with the first target entry in *makefile* if none is supplied on the command line. If, after processing its all of its dependencies, a target file is found either to be missing, or to be older than any of its dependency files, **make** rebuilds it. Optionally with this version of **make**, a target can be treated as out-of-date when the commands used to generate it have changed.

To build a given target, **make** executes the list of commands, called a *rule*. This rule may be listed explicitly in the target's makefile entry, or it may be supplied implicitly by **make**.

If no *makefile* is specified with a **–f** option:

- If there is an SCCS history file for **makefile**, **make** will attempt to extract and read the most recent version of that file.
- If there is a file named **makefile** in the current directory, **make** will attempt to read that file.
- If there is an SCCS history file for **Makefile**, **make** will attempt to extract and read the most recent version of that file.
- If there is a file named **Makefile** in the current directory, **make** will attempt to read that file.

If no *target* is specified on the command line, **make** uses the first target defined in *makefile*.

If a *target* has no makefile entry, or if its entry has no rule, **make** attempts to derive a rule by each of the following methods, in turn, until a suitable rule is found. (Each method is described under USAGE below.)

- Pattern matching rules.
- Implicit rules, read in from a user-supplied makefile.
- Standard implicit rules (also known as suffix rules), typically read in from the file **/usr/include/make/default.mk**.
- SCCS extraction. **make** extracts the most recent version from the SCCS history file (if any). See the description of the '**.SCCS_GET:**' special-function target for details.
- The rule from the '**.DEFAULT:**' target entry, if there is such an entry in the makefile.

If there is no makefile entry for a *target*, if no rule can be derived for building it, and if no file by that name is present, **make** issues an error message and stops.

## OPTIONS

**–f** *makefile*

Use the description file *makefile*. A '–' as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the standard set of implicit rules and predefined macros. When more than one '**–f** *makefile*' argument pair appears, **make** uses the concatenation of those files, in order of appearance.

**–d** Display the reasons why **make** chooses to rebuild a target; **make** displays any and all dependencies that are newer. In addition, **make** displays options read in from the **MAKEFLAGS** environment variable.

**–dd** Display the dependency check and processing in vast detail.

**–D** Display the text of the makefiles read in.

**–DD** Display the text of the makefiles, **default.mk** file, the state file, and all hidden-dependency reports.

**–e** Environment variables override assignments within makefiles.

**–i** Ignore error codes returned by commands. Equivalent to the special-function target '**.IGNORE:**'.

**–k** When a nonzero error status is returned by a command, abandon work on the current target, but continue with other branches that do not depend on that target.

**–n** No execution mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed. However, if a command line contains a reference to the **$(MAKE)** macro, that line is always executed (see the discussion of **MAKEFLAGS** in **Reading Makefiles and the Environment**).

**–p** Print out the complete set of macro definitions and target descriptions.

**–P** Report dependencies recursively to show the entire dependency hierarchy, without rebuilding any targets.

**–q** Question mode. **make** returns a zero or nonzero status code depending on whether or not the target file is up to date.

**–r** Do not read in the default file.

**–s** Silent mode. Do not print command lines before executing them. Equivalent to the special-function target '**.SILENT:**'.

**–S** Undo the effect of the **–k** option. Stop processing when a non-zero exit status is returned by a command.

**–t** Touch the target files (bringing them up to date) rather than performing their rules. *This can be dangerous when files are maintained by more than one person.* When the **.KEEP_STATE:** target appears in the makefile, this option updates the state file just as if the rules had been performed.

*macro=value*

Macro definition. This definition remains fixed for the **make** invocation. It overrides any regular definition for the specified macro within the makefile itself, or in the environment. However, this definition can still be overridden by conditional macro assignments and delayed macro assignments in target entries.

## USAGE

Refer to *Doing More with SunOS: Beginner's Guide* and **make** in *Programming Utilities and Libraries* for tutorial information about **make.**

### Reading Makefiles and the Environment

When **make** first starts, it reads the **MAKEFLAGS** environment variable to obtain a list of flag (single-character) options. Then it reads the command line for additional options that also take effect.

Next, **make** reads in a default makefile that typically contains predefined macro definitions, target entries for implicit rules, and additional rules, such as the rule for extracting SCCS files. If present, **make** uses the file **default.mk** in the current directory; otherwise it reads the file **/usr/include/make/default.mk**, which contains the standard definitions and rules. Use the directive '**include /usr/include/make/default.mk**'. in your local **default.mk** file to include them.

Next, **make** imports variables from the environment (unless the **–e** option is in effect), treating them as defined macros. Because **make** uses the most recent definition it encounters, a macro definition in the makefile normally overrides an environment variable of the same name. When **–e** is in effect, however,

environment variables are read in *after* all makefiles have been read. In that case, the environment variable takes precedence over any makefile definition.

Next, **make** reads the state file, **.make.state** in the local directory if it exists, and then any makefiles you specify with **–f**, or one of **makefile** or **Makefile** as described above.

Finally, (after reading the environment if **–e** is in effect), **make** reads in any macro definitions from the command line. These override macro definitions in the makefile and the environment both. But, if there is a definition for the macro in a makefile used by a nested **make** command, that definition takes effect for the nested **make**, unless you use the **–e** option. With **–e**, the nested **make** also uses the value set on the command line.

**make** exports its environment variables to each command or shell that it invokes. It does not export macros defined in the makefile. If an environment variable is set, and a macro with the same name is defined on the command line, **make** exports its value as defined on the command line. Unless **–e** is in effect, macro definitions within the makefile take precedence over those imported from the environment.

The macros **MAKEFLAGS**, **MAKE** and **SHELL** are special cases. See **Special-Purpose Macros** below, for details.

**Makefile Target Entries**

A target entry has the following format:

*target*... [: | ::] [*dependency*] ... [; *command*] ...
[*command*]
...

The first line contains the name of a target (or a space-separated list of target names), terminated with a colon or double colon. This may be followed by a *dependency*, or a dependency list that **make** checks in order. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise a rule for building the target.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, **make** expands macros, strips off initial TAB characters and either executes the command directly (if it contains no shell metacharacters), or passes each command line to a Bourne shell for execution.

The first line that does not begin with a TAB or # begins another target or macro definition.

**Makefile Special Characters**

*Global*

**#** Start a comment. The comment ends at the next NEWLINE. If the # follows the TAB in a command line, that line is passed to the shell (which also treats # as the start of a comment).

**include** *filename*
If the word **include** appears as the first seven letters of a line and is followed by a SPACE or TAB, the string that follows is taken as a filename to interpolate at that line. **include** files can be nested to a depth of no more than about 16.

*Targets and Dependencies*

**:** Target list terminator. Words following the colon are added to the dependency list for the target or targets. If a target is named in more than one colon-terminated target entry, the dependencies for all its entries are added to form that target's complete dependency list.

:: Target terminator for alternate dependencies. When used in place of a ':' the double-colon allows a target to be checked and updated with respect to alternate dependency lists. When the target is out-of-date with respect to dependencies listed in one entry, it is built according to the rule for that entry. When out-of-date with respect to dependencies in an alternate entry, it is built according the rule in that alternate entry. Implicit rules do not apply to double-colon targets; you must supply a rule for each entry. If no dependencies are specified, the rule is always performed.

*target* [+ *target*...] :
Target group. The rule in the target entry builds all the indicated targets as a group. It is normally performed only once per **make** run, but is checked for command dependencies every time a target in the group is encountered in the dependency scan.

% Pattern matching rule wild card character. Like the * shell wild card, % matches any string of zero or more characters occurring in both a target and the name of a dependency file. See *Pattern Matching Rules*, below for details.

*Macros*

= Macro definition. The word to the left of this character is the macro name; words to the right comprise its value. Leading white space between the = and the first word of the value is ignored. A word break following the = is implied. Trailing white space is included in the value.

$ Macro reference. The following character, or the parenthesized or bracketed string, is interpreted as a macro reference: **make** expands the reference (including the $) by replacing it with the macro's value.

( )
{ } Macro-reference name delimiters. A parenthesized or bracketed word appended to a $ is taken as the name of the macro being referred to. Without the delimiters, **make** recognizes only the first character as the macro name.

$$ A reference to the dollar-sign macro, the value of which is the character '$'. Used to pass variable expressions beginning with $ to the shell, to refer to environment variables which are expanded by the shell, or to delay processing of dynamic macros within the dependency list of a target, until that target is actually processed.

+= When used in place of '=', appends a string to a macro definition (must be surrounded by white space, unlike '=').

:= Conditional macro assignment. When preceded by a list of targets with explicit target entries, the macro definition that follows takes effect when processing only those targets, and their dependencies.

*Rules*

– **make** ignores any nonzero error code returned by a command line for which the first non-TAB character is a '–'. This character is not passed to the shell as part of the command line. **make** normally terminates when a command returns nonzero status, unless the **–i** or **–k** options, or the **.IGNORE:** special-function target is in effect.

@ If the first non-TAB character is a @, **make** does not print the command line before executing it. This character is not passed to the shell.

? Escape command-dependency checking. Command lines starting with this character are not subject to command dependency checking.

! Force command-dependency checking. Command-dependency checking is applied to command lines for which it would otherwise be suppressed. This checking is normally suppressed for lines that contain references to the '?' dynamic macro (for example, '$?').

When any combination of '–', '@', '?', or '!' appear as the first characters after the TAB, all apply. None are passed to the shell.

**Special-Function Targets**

When incorporated in a makefile, the following target names perform special-functions:

**.DEFAULT:**
If it has an entry in the makefile, the rule for this target is used to process a target when there is no other entry for it, no rule for building it, and no SCCS history file from which to extract a current version. **make** ignores any dependencies for this target.

**.DONE:** If defined in the makefile, **make** processes this target and its dependencies after all other targets are built.

**.IGNORE:**
Ignore errors. When this target appears in the makefile, **make** ignores non-zero error codes returned from commands.

**.INIT:** If defined in the makefile, this target and its dependencies are built before any other targets are processed.

**.KEEP_STATE:**
If this target appears in the makefile, **make** updates the state file, **.make.state**, in the current directory. This target also activates command dependencies, and hidden dependency checks.

**.MAKE_VERSION:**
A target-entry of the form:

**.MAKE_VERSION: VERSION**–*number*

enables version checking. If the version of **make** differs from the version indicated, **make** issues a warning message.

**.PRECIOUS:**
List of files not to delete. **make** does not remove any of the files listed as dependencies for this target when interrupted. **make** normally removes the current target when it receives an interrupt.

**.SCCS_GET:**
This target contains the rule for extracting the current version of an SCCS file from its history file. To suppress automatic extraction, add an entry for this target, with an empty rule to your makefile.

**.SILENT:**
Run silently. When this target appears in the makefile, **make** does not echo commands before executing them.

**.SUFFIXES:**
The suffixes list for selecting implicit rules (see **The Suffixes List**).

In this version of **make**, you can clear the definition of any special target, or any target starting with '.', by supplying a target entry for it with an empty rule and empty dependency list; the entry:

**.special:**

removes the definition of target named **.special**.

**Command Dependencies**

When the '**.KEEP_STATE:**' target appears in the makefile, **make** checks the command for building a target against the state file, **.make.state**. If the command has changed since the last **make** run, **make** rebuilds the target.

**Hidden Dependencies**

When the '**.KEEP_STATE:**' target appears in the makefile, **make** reads reports from **cpp**(1) and other compilation processors for any "hidden" files, such as **#include** files. If the target is out of date with respect to any of these files, **make** rebuilds it.

**Macros**

Entries of the form

*macro=value*

define macros. *macro* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value.

Subsequent references to the macro, of the forms: $(*name*) or ${*name*} are replaced by *value*. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macro definitions can contain references to other macros, in which case nested references are expanded first.

*Suffix Replacement Macro References*

Substitutions within macros can be made as follows:

$(*name*:*string1*=*string2*)

where *string1* is either a suffix, or a word to be replaced in the macro definition, and *string2* is the replacement suffix or word. Words in a macro value are separated by SPACE, TAB, and escaped NEWLINE characters.

*Pattern Replacement Macro References*

Pattern matching replacements can also be applied to macros, with a reference of the form:

$(*name*: *op*%*os*= *np*%*ns*)

where *op* is the existing (old) prefix and *os* is the existing (old) suffix, *np* and *ns* are the new prefix and new suffix, respectively, and the pattern matched by % (a string of zero or more characters), is carried forward from the value being replaced. For example:

**PROGRAM=fabricate**
**DEBUG= $(PROGRAM:%=tmp/%-g)**

sets the value of **DEBUG** to **tmp/fabricate-g**.

Note: pattern replacement macro references cannot be used in the dependency line of a pattern matching rule; the % characters are not evaluated independently.

*Appending to a Macro*

Words can be appended to macro values as follows:

*macro* += *word* ...

The space preceding the + is required. **make** inserts a leading space between the previous value and the first appended word.

**Special-Purpose Macros**

When the MAKEFLAGS variable is present in the environment, **make** takes flag (single-character) options from it, in combination with options entered on the command line. **make** retains this combined value as the MAKEFLAGS macro, and exports it automatically to each command or shell it invokes.

Note: flags passed with MAKEFLAGS are only displayed when the **-d**, or **-dd** options are in effect.

The MAKE macro is another special case. It has the value **make** by default, and temporarily overrides the **-n** option for any line in which it is referred to. This allows nested invocations of **make** written as:

$(MAKE) ...

to run recursively, with the **-n** flag in effect for all commands but **make**. This lets you use '**make -n**' to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD **make**, the MFLAGS macro is set from the MAKEFLAGS variable by prepending a '-'. MFLAGS is not exported automatically.

The **SHELL** macro, when set to a single-word value such as **/usr/bin/csh**, indicates the name of an alternate shell to use. Note: **make** executes commands containing no shell metacharacters directly. Builtin commands, such as **dirs** in the C shell, are not recognized unless the command line includes a metacharacter (for instance, a semicolon). This macro is neither imported from, nor exported to the environment, regardless of **–e**. To be sure it is set properly, you must define this macro within every makefile that requires it.

The **KEEP_STATE** environment variable, has the same effect as the '**.KEEP_STATE:**' special-function target, enabling command dependencies, hidden dependencies and writing of the state file.

**Predefined Macros**

**make** supplies the macros shown in the table that follows for compilers and their options, host architectures, and other commands. Unless these macros are read in as environment variables, their values are not exported by **make**. If you run **make** with variables by these names in the environment, it is a good idea to add commentary to the makefile to indicate what value each macro is expected to inherit from the corresponding environment variable.

If **–r** is in effect, **make** does not supply these macro definitions.

**Dynamic Macros**

There are several dynamically maintained macros that are useful as abbreviations within rules. They are shown here as references; it is best not to define them explicitly.

**$*** The basename of the current target, derived as if selected for use with an implicit rule. In the case of pattern matching rules, the value is the string matched by the '**%**'.

**$<** The name of a dependency file, derived as if selected for use with an implicit rule.

**$@** The name of the current target.

**$?** The list of dependencies that are newer than the target, derived as if selected for use with an implicit rule. Command-dependency checking is automatically suppressed for lines that contain this macro, just as if the command had been prefixed with a '**?**'. See the description of '**?**', under **Makefile Special Tokens**, above. You can force this check with the **!** command-line prefix.

**$%** The name of the library member being processed. See **Library Maintenance** for more information.

To refer to a dynamic macro within a dependency list, precede the reference with an additional '**$**' character (for example, '**$$<**'). Because **make** assigns **$<** and **$*** as it would for implicit rules (according to the suffixes list and the directory contents), they may be unreliable when used within explicit target entries.

All of these macros but **$?** can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case **F** or **D**, respectively (and enclosing the resulting name in parentheses or braces). Thus, '**$(@D)**' refers to the directory part of the string '**$@**'; if there is no directory part, '**.**' is assigned. **$(@F)** refers to the filename part.

| *Table of Predefined Macros* | | |
|---|---|---|
| *Use* | *Macro* | *Default Value* |
| *Library Archives* | AR | ar |
| | ARFLAGS | rv |
| *Assembler Commands* | AS | as |
| | ASFLAGS | |
| | COMPILE.s | $(AS) $(ASFLAGS) $(TARGET_MACH) |
| | COMPILE.S | $(CC) $(ASFLAGS) $(CPPFLAGS) $(TARGET_MACH) -c |
| *C Compiler Commands* | CC | cc |
| | CFLAGS | |
| | CPPFLAGS | |
| | COMPILE.c | $(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c |
| | LINK.c | $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) |
| *FORTRAN 77 Compiler Commands* | FC | f77 |
| | FFLAGS | |
| | COMPILE.f | $(FC) $(FFLAGS) $(TARGET_ARCH) -c |
| | LINK.f | $(FC) $(FFLAGS) $(TARGET_ARCH) $(LDFLAGS) |
| | COMPILE.F | $(FC) $(FFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c |
| | LINK.F | $(FC) $(FFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) |
| *Link Editor Command* | LD | ld |
| | LDFLAGS | |
| *lex Command* | LEX | lex |
| | LFLAGS | |
| | LEX.l | $(LEX) $(LFLAGS) -t |
| *lint Command* | LINT | lint |
| | LINTFLAGS | |
| | LINT.c | $(LINT) $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH) |
| *Modula 2 Commands* | M2C | m2c |
| | M2FLAGS | |
| | MODFLAGS | |
| | DEFFLAGS | |
| | COMPILE.def | $(M2C) $(M2FLAGS) $(DEFFLAGS) $(TARGET_ARCH) |
| | COMPILE.mod | $(M2C) $(M2FLAGS) $(MODFLAGS) $(TARGET_ARCH) |
| *Pascal Compiler Commands* | PC | pc |
| | PFLAGS | |
| | COMPILE.p | $(PC) $(PFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c |
| | LINK.p | $(PC) $(PFLAGS) $(CPPFLAGS) $(LDFLAGS) $(TARGET_ARCH) |
| *Ratfor Compilation Commands* | RFLAGS | |
| | COMPILE.r | $(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) -c |
| | LINK.r | $(FC) $(FFLAGS) $(RFLAGS) $(TARGET_ARCH) $(LDFLAGS) |
| *rm Command* | RM | rm -f |
| *yacc Command* | YACC | yacc |
| | YFLAGS | |
| | YACC.y | $(YACC) $(YFLAGS) |
| *Suffixes List* | SUFFIXES | .o .c .c~ .s .s~ .S .S~ .ln .f .f~ .F .F~ .l .l~ .mod .mod~ .sym .def .def~ .p .p~ .r .r~ .y .y~ .h .h~ .sh .sh~ .cps .cps~ |

**Implicit Rules**

When a target has no entry in the makefile, **make** attempts to determine its class (if any) and apply the rule for that class. An implicit rule describes how to build any target of a given class, from an associated dependency file. The class of a target can be determined either by a pattern, or by a suffix; there must also be a dependency file (with the same basename) from which such a target might be built. In addition to a predefined set of implicit rules, make allows you to define your own, either by pattern, or by suffix.

*Pattern Matching Rules*

A target entry of the form:

*Tp%Ts*: *Dp%Ds*
*rule*

where *Tp* is a target prefix, *Ts* is a target suffix, *Dp* is a dependency prefix, and *Ds* is a dependency suffix (any of which may be null), *pattern matching rule* in which the % stands for a basename of zero or more characters that is matched in both a filename and a dependency. When **make** encounters a match in its search for an implicit rule, it uses the rule in that target entry to build the target from the dependency file. Pattern-matching implicit rules typically make use of the $@ and $< dynamic macros as placeholders for the target and dependency names. The dynamic macro $* is set to the string matched by the % wild card.

*Suffix Rules*

When no pattern matching rule applies, **make** checks the target name to see if it contains a suffix in the known suffixes list. If so, **make** checks for any suffix rules, as well as a dependency file with same root and another recognized suffix, from which to build it.

The target entry for a suffix rule takes the form:

*DsTs*:
*rule*

where *Ts* is the suffix of the target, *Ds* is the suffix of the dependency file, and *rule* is the rule for building a target in the class. Both *Ds* and *Ts* must appear in the suffixes list.

A suffix rule with only one suffix describes how to build a target having a null (or no) suffix from a dependency file with the indicated suffix. For instance, the **.c** rule could be used to build an executable program named **file** from a C source file named '**file.c**'.

| *Table of Standard Implicit (Suffix) Rules* | | |
|---|---|---|
| *Use* | *Implicit Rule Name* | *Command Line* |
| *Assembly* | .s.o | $(COMPILE.s) –o $@ $< |
| | .s.a | $(COMPILE.s) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| *Files* | .S.o | $(COMPILE.S) –o $@ $< |
| | .S.a | $(COMPILE.S) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| *C* | .c | $(LINK.c) –o $@ $< $(LDLIBS) |
| *Files* | .c.ln | $(LINT.c) $(OUTPUT_OPTION) –i $< |
| | .c.o | $(COMPILE.c) $(OUTPUT_OPTION) $< |
| | .c.a | $(COMPILE.c) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| *FORTRAN 77* | .f | $(LINK.f) –o $@ $< $(LDLIBS) |
| *Files* | .f.o | $(COMPILE.f) $(OUTPUT_OPTION) $< |
| | .f.a | $(COMPILE.f) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| | .F | $(LINK.F) –o $@ $< $(LDLIBS) |
| | .F.o | $(COMPILE.F) $(OUTPUT_OPTION) $< |
| | .F.a | $(COMPILE.F) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |
| *lex* | .l | $(RM) $*.c<br>$(LEX.l) $< > $*.c<br>$(LINK.c) –o $@ $*.c $(LDLIBS)<br>$(RM) $*.c |
| *Files* | .l.c | $(RM) $@<br>$(LEX.l) $< > $@ |
| | .l.ln | $(RM) $*.c<br>$(LEX.l) $< > $*.c<br>$(LINT.c) –o $@ –i $*.c<br>$(RM) $*.c |
| | .l.o | $(RM) $*.c<br>$(LEX.l) $< > $*.c<br>$(COMPILE.c) –o $@ $*.c<br>$(RM) $*.c |
| *Modula 2* | .mod | $(COMPILE.mod) –o $@ –e $@ $< |
| *Files* | .mod.o | $(COMPILE.mod) –o $@ $< |
| | .def.sym | $(COMPILE.def) –o $@ $< |
| *NeWS* | .cps.h | cps $*.cps |
| *Pascal* | .p | $(LINK.p) –o $@ $< $(LDLIBS) |
| *Files* | .p.o | $(COMPILE.p) $(OUTPUT_OPTION) $< |
| *Ratfor* | .r | $(LINK.r) –o $@ $< $(LDLIBS) |
| *Files* | .r.o | $(COMPILE.r) $(OUTPUT_OPTION) $< |
| | .r.a | $(COMPILE.r) –o $% $<<br>$(AR) $(ARFLAGS) $@ $%<br>$(RM) $% |

| *Table of Standard Implicit (Suffix) Rules* | | |
|---|---|---|
| *Use* | *Implicit Rule Name* | *Command Line* |
| *Shell Scripts* | **.sh** | **cat $< >$@**<br>**chmod +x $@** |
| *yacc Files* | **.y** | **$(YACC.y) $<**<br>**$(LINK.c) –o $@ y.tab.c $(LDLIBS)**<br>**$(RM) y.tab.c** |
| | **.y.c** | **$(YACC.y) $<**<br>**mv y.tab.c $@** |
| | **.y.ln** | **$(YACC.y) $<**<br>**$(LINT.c) –o $@ –i y.tab.c**<br>**$(RM) y.tab.c** |
| | **.y.o** | **$(YACC.y) $<**<br>**$(COMPILE.c) –o $@ y.tab.c**<br>**$(RM) y.tab.c** |

**make** reads in the standard set of implicit rules from the file **/usr/include/make/default.mk**, unless **–r** is in effect, or there is a **default.mk** file in the local directory that does not **include** it.

**The Suffixes List**

The suffixes list is given as the list of dependencies for the '**.SUFFIXES:**' special-function target. The default list is contained in the SUFFIXES macro (See *Table of Predefined Macros* for the standard list of suffixes). You can define additional '**.SUFFIXES:**' targets; a **.SUFFIXES** target with no dependencies clears the list of suffixes. Order is significant within the list; **make** selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list. To place suffixes at the head of the list, clear the list and replace it with the new suffixes, followed by the default list:

**.SUFFIXES:**
**.SUFFIXES:** *suffixes* $(SUFFIXES)

A tilde ( ) indicates that if a dependency file with the indicated suffix (minus the ) is under SCCS its most recent version should be extracted, if necessary, before the target is processed.

**Library Maintenance**

A target name of the form:

*lib*(*member* ...)

refers to a member, or a space-separated list of members, in an **ar**(1V) library.

The dependency of the library member on the corresponding file must be given as an explicit entry in the makefile. This can be handled by a pattern matching rule of the form:

*lib*(%.*s*): %.**s**

where .*s* is the suffix of the member; this suffix is typically **.o** for object libraries.

A target name of the form

*lib*((*symbol*))

refers to the member of a randomized object library (see **ranlib**(1)) that defines the entry point named *symbol*.

**Command Execution**

Command lines are executed one at a time, *each by its own process or shell.* Shell commands, notably **cd**, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) just before being executed. This is suppressed if it starts with a '**@**', if there is a '**.SILENT:**' entry in the makefile, or if **make** is run with the **–s** option. Although the **–n** option specifies printing without execution, lines containing the macro $(MAKE) are executed regardless, and lines containing the **@** special character are printed. The **–t** (touch) option updates the modification date of a file without executing any rules.

This can be dangerous when sources are maintained by more than one person.

To use the Bourne shell **if** control structure for branching, use a command line of the form:

```
if expression ; \
then command ; \
        ... ; \
else ; \
        ... ; \
fi
```

Although composed of several input lines, the escaped NEWLINE characters insure that **make** treats them all as one (shell) command line.

To use the Bourne shell **for** control structure for loops, use a command line of the form:

```
for var in list ; \
        do command; \
... ; \
done
```

To write shell variables, use double dollar-signs (**$$**). This escapes expansion of the dollar-sign by **make**.

**Signals**

INT and QUIT signals received from the keyboard halt **make** and remove the target file being processed unless that target is in the dependency list for '**.PRECIOUS:**'.

## EXAMPLES

This makefile says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o -o $@
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

The following makefile uses implicit rules to express the same dependencies:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o b.o: incl.h
```

## FILES

| | |
|---|---|
| **makefile** | |
| **Makefile** | current version(s) of **make** description file |
| **SCCS/s.makefile** | |
| **SCCS/s.Makefile** | SCCS history files for the above makefile(s) |
| **default.mk** | default file for user-defined targets, macros, and implicit rules |
| **/usr/include/make/default.mk** | makefile for standard implicit rules and macros (not read if **default.mk** is) |
| **.make.state** | state file in the local directory |
| **.make.state.lock** | lock file used for controlling write access to the state file |

## SEE ALSO

**ar**(1V), **cc**(1V), **cd**(1), **get**(1), **lex**(1), **ranlib**(1), **sh**(1)

*Doing More with SunOS: Beginner's Guide*
*Programming Utilities and Libraries*

## DIAGNOSTICS

**make** returns a exit status of **1** when it halts as a result of an error. Otherwise it returns and exit status of **0**.

**Do not know how to make** *target*. **Stop.**
There is no makefile entry for *target*, and none of **make**'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

*** *target* **removed.**
**make** was interrupted while building *target*. Rather than leaving a partially-completed version that is newer than its dependencies, **make** removes the file named *target*.

*** *target* **not removed.**
**make** was interrupted while building *target* and *target* was not present in the directory.

*** *target* **could not be removed,** *reason*
**make** was interrupted while building *target*, which was not removed for the indicated reason.

**Read of include file '***file***' failed**
The makefile indicated in an **include** directive was not found, or was inaccessible.

**Loop detected when expanding macro value '***macro***'**
A reference to the macro being defined was found in the definition.

**Could not write state file '***file***'**
You used the **.KEEP_STATE:** target, but do not have write permission on the state file.

*** **Error code** *n*
The previous shell command returned a nonzero error code.

*** *signal message*
The previous shell command was aborted due to a signal. If – **core dumped** appears after the message, a **core** file was created.

## BUGS

Some commands return nonzero status inappropriately; to overcome this difficulty, prefix the offending command line in the rule with a '–'.

Filenames with the characters =, :, or @, do not work.

You cannot build **file.o** from **lib(file.o)**.

Options supplied by **MAKEFLAGS** should be reported for nested **make** commands. Use the **–d** option to find out what options the nested command picks up from **MAKEFLAGS**.

This version of **make** is incompatible in certain respects with previous versions:

- The **–d** option output is much briefer in this version. **–dd** now produces the equivalent voluminous output.
- **make** attempts to derive values for the dynamic macros '**$***', '**$<**', and '**$?**', while processing explicit targets. It uses the same method as for implicit rules; in some cases this can lead either to unexpected values, or to an empty value being assigned. (Actually, this was true for earlier versions as well, even though the documentation stated otherwise.)
- **make** no longer searches the current directory for **sccs** history files.
- Suffix replacement in macro references is now applied after the macro is expanded.

There is no guarantee that makefiles created for this version of **make** will work with earlier versions.

If there is no **default.mk** file in the current directory, and the file **/usr/include/make/default.mk** is missing, **make** stops before processing any targets. To force **make** to run anyway, create an empty **default.mk** file in the current directory.

Once a dependency is made, **make** assumes the dependency file is present for the remainder of the run. If a rule subsequently removes that file and future targets depend on its existence, unexpected errors may result.

When hidden dependency checking is in effect, the **$?** macro's value includes the names of hidden dependencies. This can lead to improper filename arguments to compiler commands when **$?** is used in a rule.

Pattern replacement macro references cannot be used in the dependency line of a pattern matching rule.

Unlike previous versions, this version of **make** strips a leading '**./**' from the value of the '**$@**' dynamic macro.

## NAME

man – display reference manual pages; find reference pages by keyword

## SYNOPSIS

**man** [ – ] [ **–t** ] [ **–M** *path* ] [ **–T** *macro-package* ] [ [ *section* ] *title* ... ] ...
**man** [ **–M** *path* ] **–k** *keyword* ...
**man** [ **–M** *path* ] **–f** *filename* ...

## DESCRIPTION

**man** displays information from the reference manuals. It can display complete manual pages that you select by *title*, or one-line summaries selected either by *keyword* (**–k**), or by the name of an associated file (**–f**).

A *section*, when given, applies to the *titles* that follow it on the command line (up to the next *section*, if any). **man** looks in the indicated section of the manual for those *title*s. *section* is either a digit (perhaps followed by a single letter indicating the type of manual page), or one of the words **new**, **local**, **old**, or **public**. If *section* is omitted, **man** searches all reference sections (giving preference to commands over functions) and prints the first manual page it finds. If no manual page is located, **man** prints an error message.

The reference page sources are typically located in the **/usr/share/man/man?** directories. Since these directories are optionally installed, they may not reside on your host; you may have to mount **/usr/share/man** from a host on which they do reside. If there are preformatted, up-to-date versions in corresponding **cat?** or **fmt?** directories, **man** simply displays or prints those versions. If the preformatted version of interest is out of date or missing, **man** reformats it prior to display. If directories for the preformatted versions are not provided, **man** reformats a page whenever it is requested; it uses a temporary file to store the formatted text during display.

If the standard output is not a terminal, or if the '–' flag is given, **man** pipes its output through **cat**(1V). Otherwise, **man** pipes its output through **more**(1) to handle paging and underlining on the screen.

## OPTIONS

**–t** **man** arranges for the specified manual pages to be **troff**ed to a suitable raster output device (see **troff**(1) or **vtroff**(1)). If both the – and **–t** flags are given, **man** updates the **troff**ed versions of each named *title* (if necessary), but does not display them.

**–M** *path*
Change the search path for manual pages. *path* is a colon-separated list of directories that contain manual page directory subtrees. For example, **/usr/share/man/u_man:/usr/share/man/a_man** makes **man** search in the standard System V locations. When used with the **–k** or **–f** options, the **–M** option must appear first. Each directory in the *path* is assumed to contain subdirectories of the form **man[1-8l-p]**.

**–T** *macro-package*
**man** uses *macro-package* rather than the standard **–man** macros defined in **/usr/share/lib/tmac/tmac.an** for formatting manual pages.

**–k** *keyword* ...
**man** prints out one-line summaries from the **whatis** database (table of contents) that contain any of the given *keyword*s.

**–f** *filename* ...
**man** attempts to locate manual pages related to any of the given *filename*s. It strips the leading pathname components from each *filename*, and then prints one-line summaries containing the resulting basename or names.

## MANUAL PAGES

Manual pages are **troff**(1)/**nroff**(1) source files prepared with the **–man** macro package. Refer to **man**(7), or *Formatting Documents* for more information.

When formatting a manual page, **man** examines the first line to determine whether it requires special processing.

**Referring to Other Manual Pages**

If the first line of the manual page is a reference to another manual page entry fitting the pattern:

.so man?*/ *sourcefile*

**man** processes the indicated file in place of the current one. The reference must be expressed as a pathname relative to to the root of the manual page directory subtree.

When the second or any subsequent line starts with **.so**, **man** ignores it; **troff**(1) or **nroff**(1) processes the request in the usual manner.

**Preprocessing Manual Pages**

If the first line is a string of the form:

'\" *X*

where *X* is separated from the the '"' by a single SPACE and consists of any combination of characters in the following list, **man** pipes its input to **troff**(1) or **nroff**(1) through the corresponding preprocessors.

| | |
|---|---|
| **e** | **eqn**(1), or **neqn** for **nroff** |
| **r** | **refer**(1) |
| **t** | **tbl**(1), and **col**(1V) for **nroff** |
| **v** | **vgrind**(1) |

If **eqn** or **neqn** is invoked, it will automatically read the file **/usr/pub/eqnchar** (see **eqnchar**(7)).

## ENVIRONMENT

| | |
|---|---|
| **MANPATH** | If set, its value overrides **/usr/share/man** as the default search path. (The **–M** flag, in turn, overrides this value.) |
| **PAGER** | A program to use for interactively delivering **man**'s output to the screen. If not set, '**more –s**' (see **more**(1)) is used. |
| **TCAT** | The name of the program to use to display **troff**ed manual pages. If not set, '**lpr –t**' (see **lpr**(1)) is used. |
| **TROFF** | The name of the formatter to use when the **–t** flag is given. If not set, **troff** is used. |

## FILES

| | |
|---|---|
| **/usr/share/man** | root of the standard manual page directory subtree |
| **/usr/share/man/man?/*** | unformatted manual entries |
| **/usr/share/man/cat?/*** | **nroff**ed manual entries |
| **/usr/share/man/fmt?/*** | **troff**ed manual entries |
| **/usr/share/man/whatis** | table of contents and keyword database |
| **/usr/share/lib/tmac/tmac.an** | standard **–man** macro package |
| **/usr/pub/eqnchar** | |

## SEE ALSO

**cat**(1V), **col**(1V), **eqn**(1), **lpr**(1), **more**(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1), **vgrind**(1), **vtroff**(1), **whatis**(1), **eqnchar**(7), **man**(7), **catman**(8)

## BUGS

The manual is supposed to be reproducible either on a phototypesetter or on an ASCII terminal. However, on a terminal some information (indicated by font changes, for instance) is necessarily lost.

Some dumb terminals cannot process the vertical motions produced by the **e** (**eqn**(1)) preprocessing flag. To prevent garbled output on these terminals, when you use **e** also use **t**, to invoke **col**(1V) implicitly. This workaround has the disadvantage of eliminating superscripts and subscripts — even on those terminals that can display them. CTRL-Q will clear a terminal that gets confused by **eqn**(1) output.

NAME

mesg – permit or deny messages on the terminal

SYNOPSIS

**mesg [ n ] [ y ]**

DESCRIPTION

**mesg** with argument **n** forbids messages with **write**(1) by revoking non-user write permission on the user's terminal. **mesg** with argument **y** reinstates permission. All by itself, **mesg** reports the current state without changing it.

FILES

**/dev/tty***

SEE ALSO

**write**(1), **talk**(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir – make a directory

SYNOPSIS

**mkdir** [ **–p** ] *dirname...*

DESCRIPTION

**mkdir** creates directories. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

The **–p** flag allows missing parent directories to be created as needed.

The current **umask**(2) setting determines the mode in which directories are created. Modes may be modified after creation by using **chmod**(1V).

**mkdir** requires write permission in the parent directory.

SEE ALSO

**chmod**(1V), **rm**(1), **mkdir**(2), **umask**(2)

NAME

mkstr – create an error message file by massaging C source files

SYNOPSIS

**mkstr** [ – ] *messagefile prefix filename*...

DESCRIPTION

**mkstr** creates files of error messages. You can use **mkstr** to make programs with large numbers of error diagnostics much smaller, and to reduce system overhead in running the program — as the error messages do not have to be constantly swapped in and out.

**mkstr** processes each of the specified *filename*s, placing a massaged version of the input file in a file with a name consisting of the specified *prefix* and the original source file name. A typical example of using **mkstr** would be:

**mkstr pistrings processed *.c**

This command would cause all the error messages from the C source files in the current directory to be placed in the file **pistrings** and processed copies of the source for these files to be placed in files whose names are prefixed with *processed*.

To process the error messages in the source to the message file, **mkstr** keys on the string '**error("**' in the input stream. Each time it occurs, the C string starting at the '**"**' is placed in the message file followed by a null character and a NEWLINE character; the null character terminates the message so it can be easily used when retrieved, the NEWLINE character makes it possible to sensibly **cat** the error message file to see its contents. The massaged copy of the input file then contains a **lseek** pointer into the file which can be used to retrieve the message, that is:

```
        char efilname[ ] = "/usr/lib/pi_strings";
        int efil = -1;

        error(a1, a2, a3, a4)
        {
                char
                buf[256];
                if (efil < 0) {
                                efil = open(efilname, 0);
                                if (efil < 0) {
oops:
                                perror (efilname);
                                exit (1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

OPTIONS

– Place error messages at the end of the specified message file for recompiling part of a large **mkstred** program.

SEE ALSO

**xstr**(1)

## NAME

more, page – browse or page through a text file

## SYNOPSIS

**more** [ **–cdflsu** ] [ *–lines* ] [ *+linenumber* ] [ *+/pattern* ] [ *filename ...* ]

**page** [ **–cdflsu** ] [ *–lines* ] [ *+linenumber* ] [ *+/pattern* ] [ *filename ...* ]

## DESCRIPTION

**more** is a filter that displays the contents of a text file on the terminal, one screenful at a time. It normally pauses after each screenful, and prints **--More--** at the bottom of the screen. **more** provides a two-line overlap between screens for continuity. If **more** is reading from a file rather than a pipe, the percentage of characters displayed so far is also shown.

**more** scrolls up to display one more line in response to a RETURN character; it displays another screenful in response to a SPACE character. Other commands are listed below.

*page* clears the screen before displaying the next screenful of text; it only provides a one-line overlap between screens.

**more** sets the terminal to *noecho* mode, so that the output can be continuous. Commands that you type do not normally show up on your terminal, except for the / and ! commands.

If the standard output is not a terminal, **more** acts just like **cat**(1V), except that a header is printed before each file in a series.

## OPTIONS

**–c** Clear before displaying. Redrawing the screen instead of scrolling for faster displays. This option is ignored if the terminal does not have the ability to clear to the end of a line.

**–d** Display error messages rather than ringing the terminal bell if an unrecognized command is used. This is helpful for inexperienced users.

**–f** Do not fold long lines. This is useful when lines contain nonprinting characters or escape sequences, such as those generated when **nroff**(1) output is piped through **ul**(1).

**–l** Do not treat FORMFEED characters (CTRL-D) as "page breaks." If **–l** is not used, **more** pauses to accept commands after any line containing a ^**L** character (CTRL-D). Also, if a file begins with a FORMFEED, the screen is cleared before the file is printed.

**–s** Squeeze. Replace multiple blank lines with a single blank line. This is helpful when viewing **nroff**(1) output, on the screen.

**–u** Suppress generation of underlining escape sequences. Normally, **more** handles underlining, such as that produced by **nroff**(1), in a manner appropriate to the terminal. If the terminal can perform underlining or has a stand-out mode, **more** supplies appropriate escape sequences as called for in the text file.

*–lines* Display the indicated number of *lines* in each screenful, rather than the default (the number of lines in the terminal screen less two).

*+linenumber*

Start up at *linenumber*.

*+/pattern*

Start up two lines above the line containing the regular expression *pattern*. Note: unlike editors, this construct should *not* end with a '/'. If it does, then the trailing slash is taken as a character in the search pattern.

## USAGE

### Environment

**more** uses the terminal's **termcap**(5) entry to determine its display characteristics, and looks in the environment variable **MORE** for any preset options. For instance, to page through files using the **–c** mode by default, set the value of this variable to **–c**. (Normally, the command sequence to set up this environment variable is placed in the **.login** or **.profile** file).

### Commands

The commands take effect immediately; it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may type the line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the '**--More--**(*xx*%)' message.

In the following commands, *i* is a numerical argument (**1** by default).

*i*SPACE Display another screenful, or *i* more lines if *i* is specified.

*i*RETURN Display another line, or *i* more lines, if specified.

*i*^**D** (CTRL–D) Display (scroll down) 11 more lines. *i* is given, the scroll size is set to *i* .

*i***d** Same as ^**D**.

*i***z** Same as SPACE, except that *i* , if present, becomes the new default number of lines per screenful.

*i***s** Skip *i* lines and then print a screenful.

*i***f** Skip *i* screenfuls and then print a screenful.

*i*^**B** (CTRL-B) Skip back *i* screenfuls and then print a screenful.

*b* Same as ^**B** (CTRL-D).

**q**
**Q** Exit from **more**.

**=** Display the current line number.

**v** Drop into the **vi**(1) editor at the current line of the current file.

**h** Help. Give a description of all the **more** commands.

*i*/*pattern* Search for the *i* th occurrence of the regular expression *pattern*. Display the screenful starting two lines prior to the line that contains the *i* th match for the regular expression *pattern*, or the end of a pipe, whichever comes first. If **more** is displaying a file and there is no such match, its position in the file remains unchanged. Regular expressions can be edited using erase and kill characters. Erasing back past the first column cancels the search command.

*i***n** Search for the *i* th occurrence of the last *pattern* entered.

Single quote. Go to the point from which the last search started. If no search has been performed in the current file, go to the beginning of the file.

!*command* Invoke a shell to execute *command* . The characters % and !, when used within *command* are replaced with the current filename and the previous shell command, respectively. If there is no current filename, % is not expanded. Prepend a backslash to these characters to escape expansion.

*i***:n** Skip to the *i* th next filename given in the command line, or to the last filename in the list if *i* is out of range.

*i***:p** Skip to the *i* th previous filename given in the command line, or to the first filename if *i* is out of range. If given while **more** is positioned within a file, go to the beginning of the file. If **more** is reading from a pipe, **more** simply rings the terminal bell.

**:f** Display the current filename and line number.

**:q**
**:Q** Exit from **more** (same as **q** or **Q**).

**.** Dot. Repeat the previous command.

**^** Halt a partial display of text. **more** stops sending output, and displays the usual **--More--** prompt. Unfortunately, some output is lost as a result.

**FILES**

| | |
|---|---|
| **/etc/termcap** | terminal data base |
| **/usr/lib/more.help** | help file |

**SEE ALSO**

**cat**(1V), **csh**(1), **man**(1), **script**(1), **sh**(1), **environ**(5V), **termcap**(5)

**BUGS**

Skipping backwards is too slow on large files.

## NAME

mt – magnetic tape control

## SYNOPSIS

**mt** [ **–f** *tapename* ] *command* [ *count* ]

## DESCRIPTION

**mt** sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable TAPE is used; if TAPE does not exist, **mt** uses the device **/dev/rmt12**. Note: *tapename* must refer to a raw (not block) tape device. By default **mt** performs the requested operation once. Operations may be performed multiple times by specifying a *count* argument.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

**mt** returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

## OPTIONS

**eof, weof**
Write *count* EOF marks at the current position on the tape.

**fsf** Forward space *count* files.

**fsr** Forward space *count* records.

**bsf** Back space *count* files.

**bsr** Back space *count* records.

For the following commands, *count* is ignored:

**rewind** Rewind the tape.

**offline, rewoffl**
Rewind the tape and place the tape unit off-line.

**status** Print status information about the tape unit.

**retension**
Wind the tape to the end of the reel and then rewind it, smoothing out the tape tension. (*count* is ignored.)

**erase** Erase the entire tape.

## FILES

| | |
|---|---|
| **/dev/rmt*** | raw magnetic tape interface |
| **/dev/rar*** | raw Archive cartridge tape interface |
| **/dev/rst*** | raw SCSI tape interface |
| **/dev/rxt*** | raw Xylogics tape interface |

## SEE ALSO

**dd**(1), **ar**(4S), **mtio**(4), **st**(4S), **environ**(5V)

## BUGS

Not all devices support all options. For example, **ar**(4S) and **st**(4S) currently do not support the **fsr, bsf,** or **bsr** options; but they are the only ones that currently support the **retension** and **rewind** options. Half-inch tapes, in particular, do not support the **retension** option.

## NAME

mv – move or rename files

## SYNOPSIS

**mv** [ – ] [ –**fi** ] *filename1 filename2*
**mv** [ – ] [ –**fi** ] *directory1 directory2*
**mv** [ – ] [ –**fi** ] [ ... *directory* ... *directory*

## DESCRIPTION

**mv** moves files and directories around in the file system. A side effect of **mv** is to rename a file or directory. The three major forms of **mv** are shown in the synopsis above.

The first form of **mv** moves (changes the name of) *filename1* to *filename2*. If *filename2* already exists, it is removed before *filename1* is moved. If *filename2* has a mode which forbids writing, **mv** prints the mode (see **chmod**(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place, otherwise **mv** exits.

The second form of **mv** moves (changes the name of) *directory1* to *directory2*, *only* if *directory2* does not already exist — if it does, the third form applies.

The third form of **mv** moves one or more *filename*s and *directories*, with their original names, into the last *directory* in the list.

**mv** refuses to move a file or directory onto itself.

## OPTIONS

– Interpret all the following arguments to **mv** as file names. This allows file names starting with minus.

–**f** Force. Override any mode restrictions and the –**i** switch. The –**f** option also suppresses any warning messages about modes which would potentially restrict overwriting.

–**i** Interactive mode. **mv** displays the name of the file or directory followed by a question mark whenever a move would replace an existing file or directory. If you type a line starting with **y**, **mv** moves the specified file or directory, otherwise **mv** does nothing with that file or directory.

## SEE ALSO

**cp**(1), **ln**(1), **chmod**(2), **rename**(2)

## BUGS

If *filename1* and *filename2* are on different file systems, then **mv** must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

**mv** will not move a directory from one file system to another.

## NAME

newgrp – log in to a new group

## SYNOPSIS

**newgrp** [ – ] [ **group** ]

## DESCRIPTION

**newgrp** changes a user's group identification. Only the group-ID is changed; the user remains a member of all groups previously established by **setgroups** (see **getgroups**(2)). The user remains logged in and the current directory is unchanged, but the group-ID of newly-created files will be set to the new effective group-ID (see **open**(2V)). The user is always given a new shell, replacing the current shell, regardless of whether **newgrp** terminated successfully or due to an error condition (such as an unknown group).

Exported variables retain their values after invoking **newgrp**; however, all unexported variables are either reset to their default value or set to null. System variables (such as **HOME, LOGNAME, PATH, SHELL, TERM,** and **USER**), unless exported by the system or explicitly exported by the user, are reset to default values. Note: the shell command **export** (see **sh**(1)) is the method to export variables, while the C shell command **setenv** (see **csh**(1)) implicitly exports its argument.

With no arguments, **newgrp** changes the group identification back to the group specified in the user's password file entry.

If the first argument to **newgrp** is a '–', the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

## FILES

| | |
|---|---|
| **/etc/group** | system group file |
| **/etc/passwd** | system password file |

## SEE ALSO

**login**(1), **su**(1), **csh**(1), **sh**(1), **open**(2V), **getgroups**(2), **initgroups**(3), **environ**(5V), **group**(5), **passwd**(5)

## BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

## NAME

nice – run a command at low priority

## SYNOPSIS

**nice** [ *–number* ] *command* [ *arguments* ]

## DESCRIPTION

There are two distinct versions of **nice**: it is built in to the C shell, and is an executable program available in **/usr/bin/nice** for use with the Bourne shell.

**nice** executes *command* with a higher "nice" value. The higher the value, the lower the command's scheduling priority. If the *number* argument is present, the nice value is incremented by that amount, up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative nice value, such as **–10**.

## FILES

**/usr/bin/nice**

## SEE ALSO

**csh**(1), **nice**(3C), **renice**(8)

## DIAGNOSTICS

**nice** returns the exit status of the subject command.

## BUGS

The **nice** C shell built-in has a slightly different syntax than the **nice** command described here. When using the built-in, the additional + option, as in:

**nice** +*n*

sets the nice value to *n* rather than incrementing by *n*.

Although you can increase the nice value for any process you own, only the super-user can decrement that value.

## NAME

nl – line numbering filter

## SYNOPSIS

**nl** [ **–p** ] [ **–h** *type* ] [ **–b** *type* ] [ **–f***type* ] [ **–v** *start* ] [ **–i** *incr* ] [ **–l** *num* ] [ **–s** *sep* ] [ **–w** *width* ] [ **–n** *fmt* ] [ **–d** *delim* ] *filename*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**nl** reads lines from *filename* (or the standard input), numbers them according to the options in effect, and sends its output to the standard output.

**nl** views the text it reads in terms of logical pages. Line numbering is normally reset at the start of each page. A logical page is composed of header, body and footer sections. The start of each page section is signaled by input lines containing section delimiters only:

*Start of file*

\:\:\:
*header*

\:\:
*body*

\:
*footer*

Empty sections are valid. Different line-numbering options are available within each section. The default scheme is no numbering for headers and footers.

## OPTIONS

**–p** Do not restart numbering at logical page delimiters.

**–b** *type* Specify which logical page body lines are to be numbered. *type* is one of:
- **a** number all lines
- **t** number lines with printable text only (the default)
- **n** , no line numbering
- **p** *rexp* number only lines that contain the regular expression *rexp*

**–h** *type* Same as **–b** *type* except for the header. The default *type* for the logical page header is **n** (no lines numbered).

**–f***type* Same as **–b** *type* except for the footer. The default for logical page footer is **n** (no lines numbered).

**–v** *start* *start* is the initial value used to number logical page lines. The default is 1.

**–i** *incr* *incr* is the increment by which to number logical page lines. The default is 1.

**–s** *sep* *sep* is the character(s) used to separate the line number from the corresponding text line. The default is a TAB.

**–w** *width* *width* is the number of characters to be used for the line-number field. The default is 6.

**–n** *fmt* **fmt** is the line numbering format. Recognized values are:
- **rn** right justified, leading zeroes suppressed (the default)
- **ln** left justified, leading zeroes suppressed
- **rz** right justified, leading zeroes kept

**–l** *num* *num* is the number of blank lines to be considered as one. For example, **–l2** results in only the second adjacent blank being numbered (if the appropriate **–ha**, **–ba**, and/or **–fa** option is set). The default is 1.

**–d** *xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **–d** and the delimiter characters. To enter a backslash, use two backslashes.

**EXAMPLE**

The command:

**nl –v10 –i10 –d!+** *filename1*

will number *filename1* starting at line number 10 with an increment of ten. The logical page delimiters are **!+**.

**SEE ALSO**

**pr**(1V)

## NAME

nm – print name list

## SYNOPSIS

**nm** [ **–gnoprsua** ] [ [ *filename* ] ...

## Sun386i SYNOPSIS

**/usr/bin/nm** [**–oxhvnefurpVT**] *filename* ...

## DESCRIPTION

**nm** prints the name list (symbol table) of each object *filename* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *filename* is given, the symbols in **a.out** are listed.

### Output Format

Each symbol name is preceded by its value (blanks if undefined) and one of the letters:

**A** absolute

**B** bss segment symbol

**C** common symbol

**D** data segment symbol

**f** filename

**t** a static function symbol

**T** text segment symbol

**U** undefined

– debug, giving symbol table entries (see **–a** below)

The type letter is upper case if the symbol is external, and lower case if it is local. The output is sorted alphabetically.

## Sun386i DESCRIPTION

The Sun386i version of the System V compatibility package includes */usr/bin/nm*, which allows the System V options to be used and creates the same output as the System V **nm**(1) command.

The System V **nm** command displays the symbol table of COFF files. *filename* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

**name** The name of the symbol.

**value** Its value expressed as an offset or an address depending on its storage class.

**class** Its storage class.

**type** Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., **char[ n ][ m ]**). Note that the object file must have been compiled with the **–g** option of the **cc**(1V) command for this information to appear.

**size** Its size in bytes, if available. (must be compiled with cc–g).

**line** The source line number at which it is defined, if available. (must be compiled with cc–g).

**section** For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

## OPTIONS

**–a** Print all symbols.

**–g** Print only global (external) symbols.

**–n** Sort numerically rather than alphabetically.

**–o** Prepend file or archive element name to each output line rather than only once.

**–p** Do not sort; print in symbol-table order.

**–r** Sort in reverse order.

**–s** Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next higher value). This difference is the value printed.

**–u** Print only undefined symbols.

**Sun386i OPTIONS**

**–o** Print the value and size of a symbol in octal instead of decimal.

**–x** Print the value and size of a symbol in hexadecimal instead of decimal.

**–h** Do not display the output header data.

**–v** Sort external symbols by value before they are printed.

**–n** Sort external symbols by name before they are printed.

**–e** Print only external and static symbols.

**–f** Produce full output. Print redundant symbols (.text, .data, .lib, and .bss), normally suppressed.

**–u** Print undefined symbols only.

**–r** Prepend the name of the object file or archive to each output line.

**–p** Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **S** (user defined segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external), the type letter is in lower case.

**–V** Print the version of the System V nm command executing on the standard error output.

**–T** By default, System V **nm** prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **–T** option causes System V **nm** to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **/usr/bin/nm** *name* **–e –v** and **/usr/bin/nm –ve** *name* print the static and external symbols in *name*, with external symbols sorted by value.

**EXAMPLE**

**example% nm**

prints the symbol list of the file named **a.out**, the default output file for the C, compiler.

**Sun386i BUGS**

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **–v** and **–n** options should be used only in conjunction with the **–e** option.

**SEE ALSO**

**ar**(1V), **as**(1), **cc**(1V), **ld**(1), **tmpnam**(3S), **a.out**(5), **ar**(5), **coff**(5)

**Sun386i DIAGNOSTICS**

**nm:** *name*: **cannot open**

if *name* cannot be read.

**nm:** *name*: **bad magic**
if *name* is not a common object file.

**nm:** *name*: **no symbols**
if the symbols have been stripped from *name*.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

**nohup** *command* [ *arguments* ]

DESCRIPTION

There are three distinct versions of **nohup**: it is built in to the C shell, and is an executable program available in **/usr/bin/nohup** and **/usr/5bin/nohup** when using the Bourne shell.

The Bourne shell version of **nohup** executes *command* such that it is immune to HUP (hangup) and TERM (terminate) signals. If the standard output is a terminal, it is redirected to the file **nohup.out**. The standard error is redirected to follow the standard output.

The priority is incremented by 5. **nohup** should be invoked from the shell with '**&**' in order to prevent it from responding to interrupts or input from the next user.

SYSTEM V DESCRIPTON

Processes run by **nohup** are immune to HUP (hangup) and QUIT (quit) signals; **nohup** does not arrange to make them immune to a TERM (terminate) signal, so unless they arrange to be immune to a TERM signal, or the shell makes them immune to a TERM signal, they will receive that signal. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**. If the standard error is a terminal, it is redirected to the standard output, otherwise it is not redirected. The priority of the process run by **nohup** is not altered.

EXAMPLE

It is frequently desirable to apply **nohup** to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell script. The command

**example% nohup sh script**

applies to everything in **script**. (If the script is to be executed often, then the need to type *sh* can be eliminated by giving **script** execute permission). Add an ampersand and the contents of **script** are run in the background with interrupts also ignored (see **sh**(1)):

**example% nohup script &**

FILES

**nohup.out**
**$HOME/nohup.out**

SEE ALSO

**chmod**(1V), **csh**(1), **nice**(1), **sh**(1), **signal**(3)

BUGS

If you use **csh**(1), then commands executed with '**&**' are automatically immune to HUP signals while in the background.

There is a C shell built-in command **nohup** that provides immunity from terminate, but does not redirect output to **nohup.out**.

**nohup** does not recognize command sequences. For instance,

**nohup** *command1* **;** *command2*

applies only to *command1* and the command:

**nohup (***command1* **;** *command2***)**

is syntactically incorrect.

Be careful of where the standard error is redirected. The following command may put error messages on tape, making it unreadable:

**nohup cpio –o <list >/dev/rmt/1m&**

while

**nohup cpio –o <list >/dev/rmt/1m 2>errors&**

puts the error messages into the file **errors.**

## NAME

nroff – format documents for display or line-printer

## SYNOPSIS

**nroff** [ **–ehig** ] [ **–m***name* ] [ **–n***N* ] [ **–o***pagelist* ] [ **–r***aN* ] [ **–s***N* ] [ **–T***name* ]

## AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**nroff** formats text in the named *files* for typewriter-like devices. See also **troff**(1). The full capabilities of **nroff** and **nroff** are described in *Formatting Documents*.

If no *file* argument is present, **nroff** reads the standard input. An argument consisting of a '–' is taken to be a file name corresponding to the standard input.

## OPTIONS

Options may appear in any order so long as they appear *before* the files.

**–e** Produce equally-spaced words in adjusted lines, using full terminal resolution.

**–h** Use output TAB characters during horizontal spacing to speed output and reduce output character count. TAB settings are assumed to be every 8 nominal character widths.

**–i** Read standard input after the input files are exhausted.

**–q** Invoke the simultaneous input-output mode of the **rd** request.

**–m***name*
Prepend the macro file **/usr/share/lib/tmac/tmac.name** to the input files.

**–n***N* Number first generated page *N*.

**–o***pagelist*
Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

**–r***aN* Set register *a* (one-character) to *N*.

**–s***N* Stop every *N* pages. **nroff** will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a NEWLINE.

**–T***name*
Prepare output for a device of the specified *name*. Known *name*s are:

| | |
|---|---|
| **37** | Teletype Corporation Model 37 terminal — this is the default. |
| **crt \| lpr \| tn300** | GE TermiNet 300, or any line printer or terminal without half-line capability. |
| **300** | DASI-300. |
| **300-12** | DASI-300 — 12-pitch. |
| **300S \| 302 \| dtc** | DASI-300S. |
| **300S-12 \| 302-12 \| dtc12** | DASI-300S. |
| **382** | DASI-382 (fancy DTC 382). |
| **382-12** | DASI-82 (fancy DTC 382 — 12-pitch). |
| **450 \| ipsi** | DASI-450 (Diablo Hyterm). |
| **450-12 \| ipsi12** | DASI-450 (Diablo Hyterm) — 12-pitch. |
| **450-12-8** | DASI-450 (Diablo Hyterm) — 12-pitch and lines-per-inch. |
| **450X** | DASI-450X (Diablo Hyterm). |
| **832** | AJ 832. |
| **833** | AJ 833. |

| | |
|---|---|
| **832-12** | AJ 832 — 12-pitch. |
| **833-12** | AJ 833 — 12-pitch. |
| **epson** | Epson FX80. |
| **itoh** | C:ITOHs0 Prowriter. |
| **itoh-12** | C:ITOHs0 Prowriter — 12-pitch. |
| **nec** | NEC 55?0s0 or NEC 77?0s0 Spinwriter. |
| **nec12** | NEC 55?0s0 or NEC 77?0s0 Spinwriter — 12-pitch. |
| **nec-t** | NEC 55?0/77?0s0 Spinwriter — Tech-Math/Times-Romanthimble. |
| **qume** | Qume Sprint — 5 or9. |
| **qume12** | Qume Sprint — 5 or 9,12-pitch. |
| **xerox** | Xerox 17?0 or Diablo 16?0. |
| **xerox12** | Xerox 17?0 or Diablo 16?0 — 12-pitch. |
| **x-ecs** | Xerox/Diablo 1730/630 — Extended Character Set. |
| **x-ecs12** | Xerox/Diablo 1730/630 — Extended Character Set, 12-pitch. |

## EXAMPLE

The following command:

**example% nroff –s4 –me users.guide**

formats **users.guide** using the **–me** macro package, and stopping every 4 pages.

## FILES

| | |
|---|---|
| **/tmp/ta*** | temporary file |
| **/usr/share/lib/tmac/tmac.*** | standard macro files |
| **/usr/share/lib/term/*** | terminal driving tables for **nroff** |
| **/usr/share/lib/term/README** | index to terminal description files |

## SEE ALSO

**checknr**(1), **eqn**(1), **tbl**(1), **troff**(1), **col**(1V), **term**(5), **man**(7), **me**(7), **ms**(7)

*Formatting Documents*

## NAME

objdump – dump selected parts of a COFF object file

## SYNOPSIS

**objdump** [ *option* [ *modifier* ... ] ] *filename* ...

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

The **objdump** command dumps selected parts of each of its object *filename* arguments. This command is compatible with System V in all its options. It will accept both object files and archives of object files.

**objdump** command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

## OPTIONS

**–a** Dump the archive header of each member of each archive file argument.

**–g** Dump the global symbols in the symbol table of an archive.

**–f** Dump each file header.

**–o** Dump each optional header.

**–h** Dump section headers.

**–s** Dump section contents.

**–r** Dump relocation information.

**–l** Dump line number information.

**–t** Dump symbol table entries.

**–z** *name* Dump line number entries for the named function.

**–c** Dump the string table.

**–L** Interpret and print the contents of the **.lib** sections.

### Modifiers

The following *modifiers* are used in conjunction with the options listed above.

**–d** *number* Dump the section number, *number*, or the range of sections starting at *number* and ending at the *number* specified by **+d**.

**+d** *number* Dump sections in the range either beginning with first section or beginning with section specified by **–d**.

**–n** *name* Dump information pertaining only to the named entity. This *modifier* applies to **–h**, **–s**, **–r**, **–l**, and **–t**.

**–p** Suppress printing of the headers.

**–t** *index* Dump only the indexed symbol table entry. The **–t** used in conjunction with **+t**, specifies a range of symbol table entries.

**+t** *index* Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **–t** option.

**–u** Underline the name of the file for emphasis.

**–v** Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be used with all the above options except **–s** and **–o**.

**–z** *name*,*number*

**–z** *name number* Dump line number entry or range of line numbers starting at *number* for the named function.

**+z** *number* Dump line numbers starting at either function *name* or *number* specified by **–z**, up to *number* specified by **+z**.

White space separating an *option* and its *modifier* is optional.

**SEE ALSO**

**coff**(5), **ar**(5)

## NAME

od – octal, decimal, hex, and ascii dump

## SYNOPSIS

**od** [ *–format* ] [ *filename* ] [ [+]*offset*[.] [**b**] [*label*] ]

## DESCRIPTION

**od** displays *file*, or its standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **–o (octal)** is the default. Dumping continues until end-of-file.

### Format Arguments

The meanings of the format argument characters are:

**a** Interpret bytes as characters and display them with their ASCII names. If the **p** character is given also, bytes with even parity are underlined. If the **P** character is given, bytes with odd parity are underlined. Otherwise the parity bit is ignored.

**b** Interpret bytes as unsigned octal.

**c** Interpret bytes as ASCII characters. Certain non-graphic characters appear as C escapes: NULL=**\0**, backspace=**\b**, formfeed=**\f**, NEWLINE=**\n**, RETURN=**\r**, TAB=**\t**; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.

**d** Interpret (short) words as unsigned decimal.

**f** Interpret long words as floating point.

**h** Interpret (short) words as unsigned hexadecimal.

**i** Interpret (short) words as signed decimal.

**l** Interpret long words as signed decimal.

**o** Interpret (short) words as unsigned octal.

**s**[*n*] Look for strings of ASCII graphic characters, terminated with a null byte. *n* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.

**v** Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an '*' in column 1.

**w**[*n*] Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.

**x** Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; if **.** is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with **x** or **0x**, it is interpreted in hexadecimal. If **b (B)** is appended, the offset is interpreted as a block count, where a block is 512 (1024) bytes. If the **file** argument is omitted, the *offset* argument must be preceded by +.

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

*label* will be interpreted as a pseudo-address for the first byte displayed. It will be shown in () following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

## SYSTEM V DESCRIPTION

The **a**, **f**, **h**, **l**, **v**, and **w** formats are not supported. The **s** format interprets (short) words as signed decimal, rather than searching for strings. The options for interpreting long or double-precision forms are not supported. The *label* argument is not supported. The **B** suffix to the *offset* argument is not supported.

SEE ALSO

**adb**(1), **dbx**(1), **dbxtool**(1)

BUGS

A file name argument can't start with +. A hexadecimal offset can't be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

## NAME

oldcompact, olduncompact, oldccat – compress and uncompress files, and cat them

## SYNOPSIS

**/usr/old/compact** [ *filename*... ]

**uncompact** [ *filename*... ]

**ccat** [ *filename*... ]

## DESCRIPTION

**Note:** This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**compact** compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted to the standard output. **compact** operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, **compact** and **uncompact** can operate as filters. In particular:

... | **compact** | **uncompact** | ...

operates as a (very slow) no-op.

When an argument **file** is given, it is compacted and the resulting file is placed in **file.C**; **file** is removed. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

**uncompact** restores the original file from a file called *file*.C which was compressed by **compact.** If no file names are given, the standard input is uncompacted to the standard output.

**ccat** cats the original file from a file compressed by **compact,** without uncompressing the file.

## FILES

| | |
|---|---|
| *.C | compacted file created by compact, removed by uncompact |

## SEE ALSO

Gallager, Robert G., *Variations on a Theme of Huffman, I.E.E.E. Transactions on Information Theory,* vol. IT-24, no. 6, November 1978, pp. 668 - 674.

## NAME

oldeyacc – modified yacc allowing much improved error recovery

## SYNOPSIS

**/usr/old/eyacc** [ **–v** ] [ *grammar* ]

## DESCRIPTION

**eyacc** is a version of **yacc**(1), that produces tables used by the Pascal system and its error recovery routines. **eyacc** fully enumerates test actions in its parser when an error token is in the look-ahead set. This prevents the parser from making undesirable reductions when an error occurs before the error is detected. The table format is different in **eyacc** than it was in the old **yacc**, as minor changes had been made for efficiency reasons.

## SEE ALSO

**yacc**(1)

*Practical LR Error Recovery* by Susan L. Graham, Charles B. Haley and W. N. Joy; SIGPLAN Conference on Compiler Construction, August 1979.

## BUGS

**pc** and its error recovery routines should be made into a library of routines for the new **yacc**.

## NAME

oldfilemerge – window-based file comparison and merging program

## SYNOPSIS

**/usr/old/filemerge** [ **–br** ] [ **–a** *ancestor* ] [ **–l** *listfile* ] [ *leftfile* [ *rightfile* [ *outfile* ] ] ]

## DESCRIPTION

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**filemerge** is a window-based version of **diff**(1), for comparing and merging text files. It displays two files for side-by-side comparison, each in a read-only text-subwindow. Beneath them, an editing subwindow can be used to construct a *merged* version—one which contains selected lines from either or both input files, along with any additional edits you may make.

*leftfile* and *rightfile* are the files to be compared, and *outfile* is name of the file containing the merged version. If *outfile* is a directory, then the output is placed in the file *outfile/leftfile*. If *outfile* is omitted, the output file is named **filemerge.out** by default. If no filename arguments are given, you can enter them from within the tool itself.

## OPTIONS

**–b** Ignore leading blanks in comparisons.

**–r** Read-only mode. Do not display the editing subwindow.

**–a** *ancestor*

Compare both files with respect to *ancestor*. A minus-sign indicates lines that have been deleted relative to the ancestor. A plus-sign indicates lines added relative to the ancestor.

**–l** *listfile*

Process a list of filename pairs. With this option, *leftfile* and *rightfile* are the names of directories, and *listfile* contains a list of filenames that appear in both. **filemerge** compares the versions of each file between the two directories, and allows you to create a merged version (typically in the directory *outfile*). The SHIFT-**Load** command button, which is selected by holding the SHIFT key while clicking on the **Load** button, reads in the next pair named in the list. If *listfile* is '-', then the list of files is read from the standard input.

## USAGE

The text in the editing subwindow (*outfile*) is initially the same as that in *leftfile*. To construct a merged version, you can directly edit the text of *outfile* with textedit commands, or you can change a selected difference to match *rightfile* (the one on the right) by clicking the **Right** button in the top panel.

### Differences

At any given time, one of the displayed "differences" is *current*. The current difference is indicated by emboldening the symbol adjacent to each line, and also by the notation "*i* **of** *n*" displayed in the control panel. Once a difference is current, you can use the **Left** and **Right** buttons to apply either the left-hand or the right-hand version of the text to *outfile*. The **Next** and **Prev** buttons select the next or previous difference, respectively.

### Property Sheet

You can customize **filemerge** using the property sheet to set or alter various display and control options. To bring up the property sheet, press the **Props** function key (typically L3) while the mouse is over any part of the **filemerge** window.

### Menus

There are pop-up menus associated with several of the control panel items, and a menu associated with the editing subwindow. The former provide to select any command function obtained with a modified mouse-button (such as SHIFT-**Next** ); the editing subwindow's menu has items that control the filename and directory location of the merged output. To bring up a menu, move the mouse-cursor to the command button, or to the editing subwindow, and hold down the RIGHT mouse-button. Select a desired menu item by releasing the mouse-button after moving the cursor on top of it.

**Command Buttons**

**Next** Make the next difference current. The subwindow scrolls, if necessary, to display it.

SHIFT-**Next** 12 Make the first difference current. (Also a menu item from the **Next** menu.)

**Prev** Make the previous difference current.

SHIFT-**Prev** 12 Make the last difference current. (Also a menu item from the **Prev** menu.)

**Right** Apply right-hand version of the current difference to *outfile*. If **autoadvance** is in effect, advance to the next difference.

SHIFT-**Right** 12 Apply the right-hand version and advance to the next difference, unless **autoadvance** is in effect. (Also a menu item from the **Right** menu.)

CTRL-**Right** Apply the right-hand version for the current difference, and for all subsequent differences up to the end of the file.

**Left** Apply the left-hand version of the current difference.

**Undo** Undo the last **Right** or **Left** operation. You can **Undo** up to 100 stacked operations. You cannot undo an **Undo**.

SHIFT-**Undo** 12 Undo all the operations since the last **Load**, or the last 100 operations.

**Scroll-Lock** When in effect, the three text-subwindows scroll in unison. Otherwise each subwindow scrolls independently.

*i* **of** *n* The number of the current difference, *i*, out of *n* detected differences. Popping up a menu on this item allows you to jump to a selected difference.

**Load** Load the files whose names appear by the prompts **File1:** and **File2:**.

SHIFT-**Load** 12 When the –l option is used, load the files from the directories shown in **File1** and **File2** corresponding to the next name in the list (taken from the *listfile* argument).

**Done** Save *outfile* and close the tool. The name used to save the file appears in the namestripe, in the same fashion as **textedit**(1).

SHIFT-**Done** 12 Save without closing. You can also save the merged version using the **Save** item in the editing subwindow's menu.

**Quit** Exit the tool. You must explicitly save your merged *outfile*, either with the **Done** button or the **Save** item in the editing subwindow's menu.

**Properties**

Hitting the L3 function key brings up a property sheet that controls several **filemerge** parameters. The information in the property sheet is stored in the file **/.filemergerc**. The property panel items have the following meanings:

**Apply** Any changes you have made to the property sheet will now take effect.

**Reset** Reset the property sheet to the state it had at the time of the last **Apply**.

**Done** Close the property sheet.

**autoadvance** Advance to the next difference after each **Left** or **Right**operation.

**Toplines** Number of lines in the top two subwindows.

**Bottomlines** Number of lines in the bottom subwindow.

**Columns** Number of columns in the left (and also right) subwindow.

FILES

| | |
|---|---|
| **/.filemergerc** | file storing property sheet information |
| **filemerge.out** | default output file |

**SEE ALSO**

**diff**(1), **sdiff**(1), **textedit**(1)

**BUGS**

Using the **Find** function key gets the subwindows out of sync for scrolling. To resync them, turn **Scroll-Lock** first off, and then on.

NAME

oldmake – maintain, update, and regenerate groups of programs

SYNOPSIS

**/usr/old/make** [ **–f** *makefile* ] ... [ **–bdeikmnpqrsSt** ] [ *target* ... ] [ *macro-name=value* ... ]

DESCRIPTION

**make** executes a list of shell commands associated with each *target*, typically to create or update a file of the same name. *makefile* contains entries for targets that describe how to bring them up to date with respect to the files and/or other targets on which each depends, called *dependencies*.

A target is out of date when the file it describes is missing, or when one (or more) of its dependency files has a more recent modification time than that of the target file. **make** recursively scans the list of dependencies for each *target* argument (or the first *target* entry in the *makefile* if no *target* argument is supplied) to generate a list of targets to check. It then checks, from the bottom up, each target against any files it depends on to see if it is out of date. If so, **make** rebuilds that target.

To rebuild a target, **make** executes the set of shell commands, called a *rule*, associated with it. This rule may be listed explicitly in a makefile entry for that target, or it may be supplied implicitly by **make**.

If no *makefile* is specified on the command line, **make** uses the first file it finds with a name from the following list:

**makefile, Makefile, s.makefile, s.Makefile, SCCS/s.makefile, SCCS/s.Makefile.**

If no *target* is specified on the command line, **make** uses the first target defined in *makefile*. If a *target* has no *makefile* entry, or if its entry has no rule, **make** attempts to update that target using an implicit rule.

OPTIONS

**–f** *makefile*
: Use the description file *makefile*. A '–' as the *makefile* argument denotes the standard input. The contents of *makefile*, when present, override the builtin rules. When more than one '**–f** *makefile*' argument pairs appear, **make** takes input from each *makefile* in the order listed (just as if they were run through **cat**(1V)).

**–b**
: This option has no effect, but is present for compatibility reasons.

**–d**
: Debug mode. Print out detailed information on files and times examined.

**–e**
: Environment variables override assignments within makefiles.

**–i**
: Ignore error codes returned by invoked commands.

**–k**
: When a nonzero error status is returned by an invoked command, abandon work on the current target but continue with other branches that do not depend on that target.

**–n**
: No execution mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed. However, if a command line contains the **$(MAKE)** macro, that line is always executed (see the discussion of **MAKEFLAGS** in **Environment Variables and Macros**).

**–p**
: Print out the complete set of macro definitions and target descriptions.

**–q**
: Question mode. **make** returns a zero or nonzero status code depending on whether or not the target file is up to date.

**–r**
: Do not use the the implicit rules **make** supplies by default. Implicit rules defined in the makefile remain in effect.

**–s**
: Silent mode. Do not print command lines before executing them.

**–S**
: Undo the effect of the **–k** option.

**–t**
: Touch the target files (bringing them up to date) rather than performing commands listed in their rules.

*macro-name=value*

Macro definition. This definition overrides any definition for the specified macro that occurs in the makefile itself, or in the environment. See **Macros** and *Environment Variables and Macros*, for details.

USAGE

Refer to *Doing More with SunOS: Beginner's Guide* and *Programming Utilities and Libraries* for tutorial information about **make**.

**Targets and Rules**

There need not be an actual file named by a target, but every dependency in the dependency list must be either the name of a file, or the name of another target.

If the target has no dependency list and no rule, or if the target has no entry in the makefile, **make** attempts to produce an entry by selecting a rule from its its set of implicit rules. If none of the implicit rules apply, **make** uses the rule specified in the .DEFAULT target (if it appears in the makefile). Otherwise **make** stops and produces an error message.

**Makefile Target Entries**

A target entry has the following format:

*target* ... : [*dependency*] ... [; *command*] ...
[*command*]
...

The first line contains the name of a target (or a space-separated list of target names), terminated with a colon (:). This may be followed by a *dependency*, or a dependency list that **make** checks in the order listed. The dependency list may be terminated with a semicolon (;), which in turn can be followed by a Bourne shell command. Subsequent lines in the target entry begin with a TAB, and contain Bourne shell commands. These commands comprise a rule for building the target, and are performed when the target is updated by **make**.

Shell commands may be continued across input lines by escaping the NEWLINE with a backslash (\). The continuing line must also start with a TAB.

To rebuild a target, **make** expands any macros, strips off initial TAB characters and passes each resulting command line to a Bourne shell for execution.

The first nonblank line that does not begin with a TAB or # begins another target or macro definition.

**Makefile Special Characters**

:: Conditional dependency branch. When used in place of a colon (:) the double-colons allow a target to be checked and updated with respect to more than one dependency list. The double-colons allow the target to have more than one branch entry in the makefile, each with a different dependency list and a different rule. **make** checks each branch, in order of appearance, to see if the target is outdated with respect to its dependency list. If so, **make** updates the target according to dependencies and rule for that branch.

\# Start a comment. The comment ends at the next NEWLINE.

$ Macro expansion. See **Macros**, below, for details.

– Following the TAB, if the first character of a command line is a '–', **make** ignores any nonzero error code it may return. **make** normally terminates when a command returns nonzero status, unless the **–i** or **–k** options are in effect.

@ Following the TAB, if the first character is a '@', **make** does not print the command line before executing it.

If '–' and '@' appear as the first two characters after the TAB, both apply.

$$ The dollar-sign, escaped from macro expansion. Can be used to pass variable expressions beginning with $ to the shell.

**Command Execution**

Command lines are executed one at a time, *each by its own shell*. Shell commands, notably **cd**, are ineffectual across an unescaped NEWLINE in the makefile. A line is printed (after macro expansion) as it is executed, unless it starts with a '**@**', there is a .SILENT entry in the dependency hierarchy of the current target, or **make** is run with the **–s** option. Although the **–n** option specifies printing without execution, lines containing the macro **$(MAKE)** are executed regardless, and lines containing the **@** special character are printed. The **–t** (touch) option updates the modification date of a file without executing any rules. This can be dangerous when sources are maintained by more than one person.

To take advantage of the Bourne shell **if** control structure for branching, use a command line of the form:

**if**
*expression*; \
**then**
*command*; \
*command*; \
...
**elif**
*expression*; \
...
**else**
*command*; \
**fi**

Although composed of several input lines, the escaped NEWLINE characters insure that **make** treats them all as one command line. To take advantage of the Bourne shell **for** control statement, use a command line of the form:

**for** *var* **in** *list* ; **do** \
*command* ; \
...
**done**

To write shell variables, use double dollar-signs (**$$**). This escapes expansion of the dollar-sign by **make**.

**Signals**

INT and QUIT signals received from the keyboard halt **make** and remove the target file being processed (unless it is in the dependency list for .PRECIOUS).

**Special-Function Targets**

When incorporated in a makefile, the following target names perform special functions.

**.DEFAULT** If this target is defined in the makefile, its rule is used when there is no entry in the makefile for a given target and none of the implicit rules applies. **make** ignores the dependency list for this target.

**.PRECIOUS** List of files not to delete. Files listed as dependencies for this target are not removed if **make** is interrupted while rebuilding them.

**.SILENT** Run silently. When this target appears in the makefile, **make** does not echo commands before executing them.

**.IGNORE** Ignore errors. When this target appears in the makefile, **make** ignores nonzero error codes returned from commands.

**.SUFFIXES** The suffixes list for selecting implicit rules (see **Implicit Rules**).

**Include Files**

**make** has an include file capability. If the word **include** appears as the first seven letters of a line, and is followed by a SPACE or a TAB, the string that follows is taken as a filename. The text of the named file is read in at the current location in the makefile. **include** files can be nested to a depth of no more than about 16.

**Macros**

Entries of the form

*macro-name=value*

define macros. *name* is the name of the macro, and *value*, which consists of all characters up to a comment character or unescaped NEWLINE, is the value. Words in a macro value are delimited by SPACE, TAB, and escaped NEWLINE characters, and the terminating NEWLINE.

Subsequent references to the macro, of the forms: $(*name*) or ${*name*} are replaced by *value*. The parentheses or brackets can be omitted in a reference to a macro with a single-character name.

Macros definitions can contain references to other macros, but the nested references aren't expanded immediately. Instead, they are expanded along with references to the macro itself.

Substitutions within macros can be made as follows:

$(*name*:*str1*=*str2*)

where *str1* is either a suffix, or a word to be replaced in the macro definition, and *str2* is the replacement suffix or word.

**Dynamically Maintained Macros**

There are several dynamically maintained macros that are useful as abbreviations within rules.

**$*** The basename of the current target. It is assigned only for implicit rules.

**$<** The name of the file on which the target is assumed to depend. This macro is only assigned for implicit rules, or within the **.DEFAULT** target's rule.

**$@** The name of the current target. It is assigned only for rules in targets that are explicitly defined in the makefile.

**$?** The list of dependencies with respect to which the target is out of date. This macro is assigned only for explicit rules.

**$%** The library member. The **$%** macro is only evaluated when the target is an archive library member of the form: *lib*(*file*.**o** ). In this case, **$@** evaluates to *lib* and **$%** evaluates to the library member, *file*.**o**.

All of these macros but **$?** can be modified to apply either to the filename part, or the directory part of the strings they stand for, by adding an upper case **F** or **D**, respectively (and enclosing the resulting name in parentheses or braces). Thus, '$(@D)' refers to the directory part of the string '$@'. If there is no directory part, '.' is generated.

**Environment Variables and Macros**

After reading in its implicit rules, **make** reads in variables from the environment, treating them as if they were macro definitions. Only then does **make** read in a makefile. Thus, macro assignments within a makefile override environment variables, provided that the **–e** option is not in effect. In turn, **make** exports environment variables to each shell it invokes. Macros not read in from the environment are *not* exported.

The **MAKEFLAGS** macro is a special case. When present as an environment variable, **make** takes its options (except for **–f**, **–p**, and **–d**) from **MAKEFLAGS** in combination with any flags entered on the command line. **make** retains this combined value, exports it automatically to each shell it forks, and reads its value to obtain options for any **make** commands it invokes. Note, however that flags passed with **MAKEFLAGS** even though they are in effect, are not shown in the output produced by **make**.

The **MAKE** macro is another special case. It has the value **make** by default, and temporarily overrides the **–n** option for any line that contains a reference to it. This allows nested invocations of **make** written as:

**$(MAKE)** ...

to run recursively, so that the command **make –n** can be used to test an entire hierarchy of makefiles.

For compatibility with the 4.2 BSD **make**, the MFLAGS macro is set from the MAKEFLAGS variable by prepending a '–'. MFLAGS is not exported automatically.

**make** supplies the following macros for compilers and their options:

| | | | |
|---|---|---|---|
| CC | C compiler, **cc**(1V) | CFLAGS | C compiler options |
| FC | FORTRAN 77 compiler, **f77**(1) | FFLAGS | FORTRAN 77 compiler options |
| | | RFLAGS | FORTRAN 77 compiler options with Ratfor (**.r**) source files |
| PC | Pascal compiler, **pc**(1) | PFLAGS | Pascal compiler options |
| M2C | Modula-2 compiler | M2FLAGS | Modula-2 compiler options |
| GET | **sccs**(1) **get** command | GFLAGS | **sccs get** options |
| AS | the assembler, **as**(1) | ASFLAGS | assembler options |
| LD | the linker, **ld**(1) | LDFLAGS | linker options |
| LEX | **lex**(1) | LFLAGS | **lex** options |
| YACC | **yacc**(1) | YFLAGS | **yacc** options |

Unless these macros are read in as environment variables, their values are not exported by **make**. If you run **make** with any these set in the environment, it is a good idea to add commentary to the makefile to indicate what value each takes. If **–r** is in effect, **make** ignores these macro definitions.

When set to a single-word value such as **/usr/bin/csh**, the SHELL macro indicates the name of an alternate shell to use for invoking commands. Note: to improve normal performance **make** executes command lines that contain no shell metacharacters directly. Such builtin commands as **dirs**, or **set** in the C shell are not recognized unless the command line includes a metacharacter (for instance, a semicolon).

**Implicit Rules**

**make** supplies implicit rules for certain types of targets that have no explicit rule defined in the makefile. For these types of targets, **make** attempts to select an implicit rule by looking for an association between the target and a file in the directory that shares its basename. That file, if found, is presumed to be a dependency file. The implicit rule is selected according to the target's suffix (which may be null), and that of the dependency file. If there is no such dependency file, if the suffix of either dependency or target is not the suffixes list, or if there is no implicit rule defined for that pair of suffixes, no rule is selected. **make** either uses the default rule that you have supplied (if any), or stops.

The suffixes list is a target with each known suffix listed as a dependency, by default:

**.SUFFIXES: .o .c .c~ .mod .mod~ .sym .def .def~ .p .p~ .f .f~ .r .r~ .y .y~ .l .l~**
**.s .s~ .sh .sh~ .h .h~**

Multiple suffix-list targets accumulate; a **.SUFFIXES** target with no dependencies clears the list of suffixes. Order is significant; **make** selects a rule that corresponds to the target's suffix and the first dependency-file suffix found in the list.

A tilde (~) refers to the **s.***prefix* of an SCCS history file (see **sccs**(1)). If **make** cannot locate a history file (with a name of the form **s.***basename.suffix*) in the current working directory, it checks for one in the SCCS subdirectory (if that directory exists) for one from which to **get**(1) the dependency file.

An implicit rule is a target of the form:

*dt*:
    *rule*

where *t* is the suffix of the target, *d* is the suffix of the dependency, and *rule* is the implicit rule for building such a target from such a dependency. Both *d* and *t* must appear in the suffixes list for **make** to recognize the target as one that defines an implicit rule.

An implicit rule with only one suffix describes how to build a target having a null (or no) suffix, from a dependency having the indicated suffix. For instance, the **.c** rule describes how to build the executable *file* from a C source file, *file***.c**.

Implicit rules are supplied for the following suffixes and suffix pairs:

**.c .c~ .p .p~ .mod .mod~ .f .f~ .F .F~ .r .r~ .sh .sh~ .c.o .c~.o .c~.c .p.o .p~.o .p~.p .mod.o .mod~.o .mod~.mod .def.sym .def~.sym .def~.def .f.o .f~.f .F.o .F~.o .F~.F .r.o .r~.o .r~.r .s.o .s~.o .s~.s .sh~.sh .y.o .y~.o .l.o .l~.o .y.c .y~.c .y~.y .l.c .l~.c .l~.l .c.a .c~.a .s~.a .h~.h**

These rules can be changed within a makefile, and additional implicit rules can be added. To print out **make**'s internal rules, use the following command. Note: this command only works with the Bourne Shell:

```
$ make -fp - 2>/dev/null </dev/null
```

If you are using the C shell, use this command to print out **make**'s internal rules:

```
example% (make -fp - </dev/null >/dev/tty) >&/dev/null
```

**Library Maintenance**

If a target name contains parentheses, as with:

**lib.a(member)**

it is assumed to be the name of an archive (**ar**(1V)) library. The string within the parentheses refers to a member of the library. (If the string contains more than one word, the only first word is used.) A member of an archive can be explicitly made to depend on a file with a matching filename. For instance, given a directory that contains the files **mem1.c** and **mem2.c**, along with a makefile with the entries:

```
lib.a: lib.a(mem1.o) lib.a(mem2.0)

lib.a(mem1.o): mem1.o
        ar rv lib.a mem1.o

lib.a(mem2.o): mem2.o
        ar rv lib.a mem2.o
```

**make**, when run, compiles the **.c** files into relocatable object (**.o**) files using the **.c.o** implicit rule. It then loads the freshly compiled version of each file into the library according to the explicit rules in the **lib.a()**targets.

Implicit rules pertaining to archive libraries have the form *.XX*.**a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires that *XX* to be different from the suffix of the archive member itself. For instance, the target **lib**(*file*.**o**) cannot depend upon the *file*.**o** explicitly, but instead, must be made to depend on a source file, such as *file*.**c**. For this reason it is recommended that you define an explicit target in the makefile for each library member to maintain, as shown above.

A target name of the form

*library*((*entry-point*))

refers to the member of a randomized object library (see **ranlib**(1)) that defines the symbol *entry-point*.

**EXAMPLES**

This *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) along with a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o -o $@

a.o: incl.h a.c
        cc -c a.c

b.o: incl.h b.c
        cc -c b.c
```

The following *makefile* uses the builtin inference rules to express the same dependencies:

```
pgm: a.o b.o
        cc a.o b.o -o pgm

a.o b.o: incl.h
```

FILES

*[Mm]akefile*
s.*[Mm]akefile*
SCCS/s.*[mM]akefile*
**/usr/bin/csh**

DIAGNOSTICS

**Don't know how to make** *target* **. Stop.**

There is no makefile entry for *target*, and none of **make**'s implicit rules apply (there is no dependency file with a suffix in the suffixes list, or the target's suffix is not in the list).

*** *target* **removed.**

**make** was interrupted in the middle of trying to build *target*. Rather than leaving a partially-completed version that is newer than its dependencies, make removes the file associated with *target*.

*** **Error code** *n*.

The previous shell command returned a nonzero error code. In this case *make* stops, unless either the **–k** or the **–i** option is set, the target **.IGNORE** appears, or the command is prefixed with a '–' in the makefile.

*** *signal message*

The previous shell command was aborted due to a signal. If '– **core dumped**' appears after the message, a **core** file was created.

SEE ALSO

**ar**(1V), **cat**(1V), **cc**(1V), **cd**(1), **csh**(1), **get**(1), **lex**(1), **ranlib**(1), **sccs**(1), **sh**(1)

*Doing More with SunOS: Beginner's Guide*
*Programming Utilities and Libraries*

BUGS

Some commands return nonzero status inappropriately; use **–i** to overcome the difficulty.

Filenames with the characters =, :, and @ will not work.

You cannot build **lib**(*file*.**o**) from *file*.**o**.

The macro substitution **$(a:.o=.c )** does not work.

Options supplied by **MAKEFLAGS** should appear in output from **make**.

**NAME**

oldprmail – display waiting mail

**SYNOPSIS**

**/usr/old/prmail** [ *user* ] ...

**DESCRIPTION**

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**prmail** displays waiting mail for you, or the specified *user*s. The mail is not disturbed.

**FILES**

**/var/spool/mail/*** waiting mail files

**SEE ALSO**

**biff(1), mail(1), from**(1), **binmail(1)**

## NAME

oldpti – phototypesetter interpreter

## SYNOPSIS

**/usr/old/pti** [ *filename* ... ]

## DESCRIPTION

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**pti** shows the commands in a stream from the standard output of **troff**(1) using **troff**'s **–t** option, interpreting the commands as they would act on the typesetter. Horizontal motions show as counts in internal units and are marked with < and > indicating left and right motion. Vertical space is called *leading* and is also indicated.

The output is really cryptic unless you are an experienced C/A/T hardware person. It is better to use '**troff –a**'.

## SEE ALSO

**troff**(1)

**NAME**

oldsetkeys – modify interpretation of the keyboard

**SYNOPSIS**

**/usr/old/setkeys [ reset | nosunview | [ [ lefty ] [ noarrows ] ] ] [ sun1 | sun2 | sun3 ]**

**Sun386i SYNOPSIS**

**setkeys [ reset | nosunview | [ [ lefty ] [ noarrows ] ] ] [ sun1 | sun2 | sun3 ]**

**DESCRIPTION**

**setkeys** has been superseded by the **Input** category in **defaultsedit**(1), and by the program **input_from_defaults** (1). It is retained for backwards compatibility on Sun-2, Sun-3 and Sun-4 systems.

**Sun386i DESCRIPTION**

**setkeys** changes the kernel's keyboard translation tables, converting a keyboard to one of a number of commonly desired configurations. It takes an indication of the modifications to be performed, and optionally, the kind of keyboard attached to the user's machine. It affects all keyboard input for the machine it is run on (in or out of the window system) until that effect is superseded by rebooting, or by running '**setkeys reset**'.

**OPTIONS**

*modifications*

Empty, or one of **reset, nosunview**, or some combination of **lefty** and **noarrows**. By default, the keyboard is set to produce the SunView function-key codes (**Stop, Props, Front, Close, Find, Again, Undo, Copy, Paste** and **Cut**; *SunView 1 Beginner's Guide*. On Sun2 and Sun3 keyboards, this is meaningless; on the Sun1, those functions are assigned to two columns of the right numeric-function pad.

**Lefty** Indicate the SunView functions are to be produced from keys on the right side of the keyboard, convenient for using the mouse in the left hand.

On the Sun2 and Sun3 systems, the SunView functions are reflected to the outside columns of the right function pad; those right-side functions are distributed in a more complicated fashion dictated by keeping the arrow keys together; see below. Also, the Line Feed key, immediately below Return, is converted to a second Control.

On the Sun1, **Lefty** is the same as the default, since there is no left function pad.

**Noarrows**

Reassign the keys with cursor arrows on their caps to produce simple function codes (so they may be used with filters in the textsw, or mapped input in the ttysw).

**Nosunview**

Valid only on a Sun2 or Sun3 keyboard, and incompatible with **Lefty, Noarrows**, or **Reset**. This option assigns new codes to keys F1 and L2 - L10, codes that are not normally produced anywhere on the keyboard. These codes may be selected by a *mapi* or *mapo* operation defined in a user's **.ttyswrc** file.

This option supports a measure of backwards compatibility to programs that apply some other interpretation to the affected function keys. It allows them to access the new codes when the standard codes would be preempted by SunView functions (for instance, in a **tty**(1) subwindow).

**Reset** Incompatible with **Lefty, Noarrows**, or **Nosunview**; it causes the keyboard to be reset to its original interpretation.

*keyboard-type*

One of **sun1, sun2**, or **sun3**. Normally, this option is omitted; the type of keyboard attached to the system is obtained from the kernel. If included, the option is believed in preference to the kernel's information. **setkeys** treats Sun2 and Sun3 keyboards identically except when the modification is **Reset**.

Note: the keyboard type is not necessarily the same as the machine type. A Sun1 keyboard is the

VT100-style keyboard shipped with Model 100Us, while Sun2 and Sun3 keyboards may be attached interchangeably to Sun-2 and Sun-3 machines. A Sun3 keyboard is distinguished by its aerodynamic housing, and the presence of Caps and Alternate keys.

Options may appear in any order, and case is not significant. The accompanying diagrams show the exact distribution of codes for each combination of keyboard and arguments to setkeys.

## EXAMPLES

The command

**setkeys lefty noarrows**

puts the SunView functions on the right pad of the keyboard, replacing arrow keys by the corresponding right-function codes, and displacing right-function codes to the left pad.

The command:

**setkeys sun1 reset**

restores a Sun1 keyboard to its original arrangement.

## SEE ALSO

**defaultsedit**(1), **input_from_defaults**(1), **kb**(4M)

*SunView 1 Beginner's Guide*

## BUGS

**setkeys** affects the kernel's key tables, which in turn affects all users logged in to the system.

## DIAGRAMS

**Sun1, reset:**

```
            ^  V  <  >
[       standard        ]     TF1   TF2   TF3   TF4
[        typing         ]     7     8     9     -
[        array          ]     4     5     6     ,
[         ....          ]     1     2     3     En-
                               0           .     ter
```

**default / lefty:**

```
            ^  V  <  >
[       standard        ]     Again RF1   Stop  RF2
[        typing         ]     Undo  RF3   Props RF4
[        array          ]     Put   RF5   Front RF6
[         ....          ]     Get   RF7   Close RF8
                               Delete Find
```

**default / lefty, noarrow:**

```
             TF1 TF2 TF3 TF4
[       standard        ]     Again RF1   Stop  RF2
[        typing         ]     Undo  RF3   Props RF4
[        array          ]     Put   RF5   Front RF6
[         ....          ]     Get   RF7   Close RF8
                               Delete Find
```

**Sun2 & Sun3,**

**reset / default:**

| | | | | | | |
|---|---|---|---|---|---|---|
| | | TF1 TF2 ... | ] | | | |
| Stop | Again | [ standard | ] | RF1 | RF2 | RF3 |
| Props | Undo | [ typing | ] | RF4 | RF5 | RF6 |
| Front | Put | [ array | ] | RF7 | ^ | RF9 |
| Close | Get | [ | Retn | < | RF11 | > |
| Find | Delete | [ | LF | RF13 | V | RF15 |

**noarrows (only):**

| | | | | | | |
|---|---|---|---|---|---|---|
| | | TF1 TF2 ... | ] | | | |
| Stop | Again | [ standard | ] | RF1 | RF2 | RF3 |
| Props | Undo | [ typing | ] | RF4 | RF5 | RF6 |
| Front | Put | [ array | ] | RF7 | RF8 | RF9 |
| Close | Get | [ | Retn | RF10 | RF11 | RF12 |
| Find | Delete | [ | LF | RF13 | RF14 | RF15 |

**lefty:**

| | | | | | | |
|---|---|---|---|---|---|---|
| | | TF1 TF2 ... | ] | | | |
| Stop | RF1 | [ standard | ] | Again | < | Stop |
| RF6 | RF4 | [ typing | ] | Undo | > | Props |
| RF9 | RF7 | [ array | ] | Put | ^ | Front |
| RF12 | RF10 | [ | Retn | Get | RF11 | Close |
| RF15 | RF13 | [ | Ctrl | Delete | V | Find |

**lefty, noarrows**

| | | | | | | |
|---|---|---|---|---|---|---|
| | | TF1 TF2 ... | ] | | | |
| Stop | RF1 | [ standard | ] | Again | RF2 | Stop |
| RF6 | RF4 | [ typing | ] | Undo | RF5 | Props |
| RF9 | RF7 | [ array | ] | Put | RF8 | Front |
| RF12 | RF10 | [ | Ret | Get | RF11 | Close |
| RF15 | RF13 | [ | Ctrl | Delete | RF14 | Find |

**nosunview:**

| | | | | | | |
|---|---|---|---|---|---|---|
| | | LF11 TF2 ... | ] | | | |
| Stop | TF11 | [ standard | ] | RF1 | RF2 | RF3 |
| LF12 | TF12 | [ typing | ] | RF4 | RF5 | RF6 |
| LF13 | TF13 | [ array | ] | RF7 | ^ | RF9 |
| LF14 | TF14 | [ | Ret | < | RF11 | > |
| LF15 | TF15 | [ | LF | RF13 | V | RF15 |

## NAME

oldsun3cvt – convert large Sun-2 system executables to Sun-3 system executables

## SYNOPSIS

**/usr/old/sun3cvt** [ *oldfile* [ *newfile* ] ]

## DESCRIPTION

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**sun3cvt** converts an old Sun-2 system program file (predating Sun release 3.0) into a Sun-3 system executable file.

The default *oldfile* is **a.out**. The default *newfile* is **sun3.out**.

**sun3cvt** attempts to create a file of the same type (magic number). However, for sharable-text files with less than 128kb of text, the new file will not be sharable (since Sun-3 data segments must begin on 128kb boundaries). Also, most programs have some text in the data segment as a consequence of the new larger Sun-3 system page and segment sizes.

**execve**(2) executes an old Sun-2 system program file, but the program's text is not sharable. Old pure-text programs with text segments larger than 128kb can be made sharable on machines running release 3.0 or subsequent releases by using **sun3cvt**.

## FILES

| | |
|---|---|
| **a.out** | default *oldfile* |
| **sun3.out** | default *newfile* |

## SEE ALSO

**execve**(2)

## NAME

oldsyslog – make a system log entry

## SYNOPSIS

**/usr/old/syslog** [ – ] [ **–p** ] [ **–i** *name* ] [ *–level* ] [ message . . . ]

## DESCRIPTION

Note: This program is considered to be obsolete, and will not be distributed or supported in future Sun releases.

**syslog** sends the specified message (or *stdin* if '–' is specified) as a system log entry to the syslog daemon. The log entry is sent to the daemon on the machine specified by the *loghost* entry in the **/etc/hosts** file.

## OPTIONS

– Each line of the standard input is sent as a log entry.

**–p** **syslog** will log its process ID in addition to the other information.

**–i** *name* The specified name will be used as the " ident " for the log entry.

*–level* The message will be logged at the specified level. The level can be specified numerically, in the range 1 through 9, or symbolically using the names specified in the include file **/usr/include/syslog.h** (with the leading **LOG_** stripped off). **syslog-HELP** will list the valid symbolic level names. Only the super-user can make log entries at levels less than or equal to **SALERT.**

## FILES

| | |
|---|---|
| **/usr/etc/syslogd** | syslog daemon |
| **/usr/include/syslog.h** | names of logging levels |

## SEE ALSO

**syslog**(3), **syslogd**(8)

NAME

oldtektool – SunView Tektronix 4014 terminal-emulator window

SYNOPSIS

**/usr/old/tektool** [ **–s[lcdeg[ce]m[12]]** ] [ **–[cr]** *command-line* ] [ **–f** *fontdir* ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**tektool** emulates a Tektronix 4014 terminal with the enhanced graphics module. It does this in much the same way as shelltool (see **shelltool**(1)) emulates a regular glass tty. When **tektool** is invoked, a command (usually a shell) is started up, its output and input are connected to the emulator, and a new window is formed. The default window is the entire screen. When the emulator is running, keys TF(1) through TF(3), (usually function keys F1-F3 (see **kbd**(4S)) have special meaning.

TF (1) Unshifted, this is the 4014 **PAGE** button. Shifted, this is the 4014 **RESET** button.

TF (2) Copy screen. The raster image (**/usr/include/rasterfile.h**) of the 4014 screen is piped to a command found in the **TEKCOPY** environment variable. If **TEKCOPY** is not found in the environment, '**lpr–v**' is used. The copy button is unaffected by window manipulations, and will transmit the contents of the 4014 screen only.

TF (3) Release page full condition.

These functions are also available through the tool menu. When in graphics input (GIN) mode and the 4014 crosshairs are visible, the left hand mouse button may be used as the space bar to terminate GIN mode.

OPTIONS

**–s** Specifies the Tektronix 4014 strap options with the following modifiers:

l Received LINEFEED characters also generate RETURN characters.
c Received RETURN characters also generate LINEFEED characters.
d Received DELETE characters are used as low order Y axis ( LOY ) addresses.
e Echo keyboard input.
g Graphic input mode (GIN) terminator specification. If this strap is followed by a **c**, GIN mode data is terminated by a RETURN character. If it is followed by a **e**, GIN mode data is terminated by a RETURN character followed by an EOT character. If this strap is not present, no characters are sent after GIN mode data.
m Page full control specification. If this strap is followed by a **1**, **tektool** will stop accepting tty input when a LINEFEED character is done past the last line in margin 1. This is the 4014 page full condition. The page full condition it released by a **PAGE** or a **RELEASE** or any ASCII keyboard input. If this strap is followed by a **2**, the page full condition happens at the end of margin 2. If this strap is not present, the page full condition never occurs.

If the **–s** option is not given, the environment is searched for the **TEKSTRAPS** variable which provides the modifiers. The straps may also be set by the property sheet available by selecting the **PROPS** menu item. If no straps are specified the **d** strap is assumed. **–f** *fontdir* Look for fonts in the directory specified by *fontdir*. The fonts must be called **tekfont**0 through **tekfont**3. Fonts must be in **vfont**(5) format. If this option is not given, the font directory is obtained from the TEKFONTS environment variable (if it exists). If no font directory is specified, **/usr/lib/fonts/tekfonts** is used.

**–c** *command-line*

Take terminal emulator input from a shell which in turn runs the *command-line* following the **–c** option.

**–r** *command-line*

Run *command-line* to provide input to the terminal emulator. This must be the last option, since the remainder of the arguments are used by the command.

**FILES**

**/usr/lib/fonts/tekfonts/tekfont[0-3]**

**SEE ALSO**

**sunview**(1), **shelltool**(1), **kbd**(4S), **vfont**(5)

*Installing the SunOS*

**BUGS**

Special point plot mode is not supported.

Z-axis stuff, except for defocusing, is not supported.

Defocused alpha characters are not supported.

**DIAGNOSTICS**

**copy command failed** The copy command in the **TEKCOPY** environment variable or in the property sheet could not be executed.

**CAVEATS**

Like all 4014 emulators, this does not duplicate every nuance of the 4014. For instance, certain programs redraw stuff already on the screen in order to highlight things with the storage flash. This will not work here. Also, even though the emulator supports the full 4096 point addressing of the 4014, it cannot display this on the screen. All points will be rounded to the nearest available pixel. This may cause some funny effects.

The **tektool** window may be treated just like other windows; it can be overlaid, moved, reshaped etc. However, when the window is reshaped, the contents will not scale.

## NAME

oldvc – version control

## SYNOPSIS

**/usr/old/vc** [ **–a** ] [ **–s** ] [ **–t** ] [ **–c***c* ] [ *keyword=value*]...

## DESCRIPTION

The **vc** command copies lines from the standard input to the standard output under control of its *arguments* and the *control statements* encountered in the standard input. In the process of performing the copy operation, a reference to a user-declared *keyword* is replaced by its character-string *value*, when appearing in a plain text line or control statement.

Copying lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified either in control statements, or as **vc** command-line arguments.

## OPTIONS

**–a** Force replacement of keyword references in *all* text lines, and not just in **vc** statements.

**–s** Silent. Suppress warning messages (but not error messages) that are normally printed on the diagnostic output.

**–t** If a TAB character appears on a line, all characters from the beginning of a line, up to and including the first TAB, are ignored for the purpose of detecting a control statement. If the TAB precedes a control statement, the leading text is discarded.

**–c***c* Specify an alternate control character to use instead of ':'.

## USAGE

**vc** distinguishes between text input lines and version control statement lines. A version control statement (control statement) is a single line beginning with a control character. The default control character is colon (:), except as modified by the **–c***c* option. Input lines beginning with a backslash (\), and followed by a control character, are not control lines and are copied to the standard output as text with the backslash removed. Lines beginning with a backslash, but not followed by a control character, are copied in their entirety.

### Keyword Replacement

A keyword is composed of 9 or less alphanumeric characters; the first must be alphabetic. A value is any printable ASCII character or character string. An unsigned string of digits is treated as a numeric value in control operations. Keyword values may not contain any SPACE or TAB characters.

Keyword replacement is performed whenever a keyword, surrounded by control characters, is encountered on a version control statement. The **–a** option forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with '\'. If a literal '\' is desired, then it too must be preceded by a '\'.

### Version Control Statements

**: dcl** *keyword*[,*keyword*]

Declare a keyword. All keywords must be declared.

**: asg** *keyword=value*

Assign a value to a keyword. An **asg** statement overrides any previous assignment for the corresponding keyword, including those on the **vc** command line. Keywords declared, but not assigned values, have null values.

**:if** [**not**] *condition*

...

**:end** Skip lines of the standard input. When *condition* is TRUE, all lines between the **if** statement and the matching **end** statement are *copied* to the standard output. Otherwise, the intervening lines are discarded and ignored, *including intervening control statements.* Intervening **if** and **end** control statements are recognized solely for the purpose of maintaining the proper **if**–**end** matching. The **not** argument inverts the sense of the condition. When it is used, intervening lines are included in

the output only when the conditions is false.

*condition* is a logical expression composed of *comparisons* and logical operators. A *comparison* consists of two text values (may be keyword references) separated by a comparison operator. Each value must be separated from all operators by at least one SPACE . Numeric strings are treated as unsigned integers for certain comparisons. The comparison operators are:

| | |
|---|---|
| = | equal (string) |
| != | not equal (string) |
| > | greater than (numeric) |
| < | less than (numeric) |

For instance, the line:

**:if** *xxx* != *yyy*

tests to see whether the string '*xxx*' is not equal to '*yyy*', which is true; subsequent intervening lines are therefore included.

The logical sense of comparisons can be combined using the logical operators (in order of precedence):

| | |
|---|---|
| ( ) | logical grouping |
| & | and |
| \| | or |

For instance, the line

**:if** *xxx* = *yyy* | *xxx* != *yyy*

is true because either comparison will make it true, while

**:if** *xxx* = *yyy* & *xxx* != *yyy*

is false, because in this case, both comparisons must be true.

**::***text* Force keyword replacement on lines that are copied to the standard output, independent from the **–a** option. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file.

**:on**
**:off** Turn on or off keyword replacement on all lines.

**:ctl** *c* Change the control character to *c*.

**:msg** *message*
Print *message* on the diagnostic output.

**:err** *message*
Print the given message, followed by:

**ERROR: err statement on line ... (915)**

on the diagnostic output; vc halts execution, and returns an exit code of **1**.

SEE ALSO
**help**(1)

DIAGNOSTICS
Use **help**(1) for explanations.

## NAME

on – execute a command on a remote system, but with the local environment

## SYNOPSIS

**on** [ **–i** ] [ **–d** ] [ **–n** ] *host command* [ *argument* ] . . .

## DESCRIPTION

The **on** program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

The standard input is connected to the standard input of the remote command, and the standard output and the standard error from the remote command are sent to the corresponding files for the **on** command.

## OPTIONS

**–i** Interactive mode. Use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated.

**–d** Debug mode. Print out some messages as work is being done.

**–n** No Input. This option causes the remote program to get EOF when it reads from the standard input, instead of passing the standard input from the standard input of the **on** program. For example, **–n** is necessary when running commands in the background with job control.

## SEE ALSO

**exports**(5), **rexd**(8C)

## DIAGNOSTICS

**unknown host** Host name not found.

**cannot connect to server**
Host down or not running the server.

**can't find** Problem finding the working directory.

**can't locate mount point**
Problem finding current file system.

Other error messages may be passed back from the server.

## BUGS

The SunView window system can get confused by the environment variables.

When the working directory is remote mounted over NFS, a ^**Z** hangs the window.

NAME

organizer – file and directory manager

SYNOPSIS

**organizer**

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**organizer** is a SunView application for viewing and manipulating files and directories. It performs many of the functions of the **ls, cd, cp, rm, mv, mkdir, rmdir, backup, restore, find,** and **chmod** commands with a visual interface.

At any given time, the **organizer** window normally shows the files and directories in a single directory, representing each file or directory with an appropriate illustrated icon. The illustration indicates whether a file is a directory, contains text, is an executable program, or optionally a user-defined file type.

When **organizer** is switched into Map mode, the icons are arranged to indicate the hierarchy of files and directories. Double clicking on a directory icon shows the contents of that directory in a new column.

Several display modes are available, and can be set for an individual **organizer** window or for all **organizer** windows. You can select whether hidden files are shown, whether just the name, the name and information, or name and icon are shown for each file and directory, and how the contents are sorted.

Text files can be "opened" by double clicking on the file's icon. The contents of the file are then shown and can be edited in a separate text editor window. Double-clicking always opens files; in user-defined file types, you can specify the OPEN, EDIT, and PRINT applications.

You can move down through the directory hierarchy by double clicking on a directory icon, and up by double clicking on the up arrow button on the command panel.

Copying, moving, and deleting require you to select one or more files. To select a file, click the left button on it (don't double click—this will open the file). To select additional files to be operated on, click the middle button on each additional file. Copying and moving operations require a destination directory. After the files are selected, change directories to the desired destination as described above, and then "drop" the files with the Drop button on the command panel. If the copy involves overwriting an identically named file, **organizer** will prepend **copy_of_** to the filename of the new file.

FILES

**/usr/include/images/*** file and directory icons

## NAME

overview – run a program from SunView that takes over the screen

## SYNOPSIS

**overview** [ **–w** ] [ *generic_tool_flags* ] *program_name* [ *arguments* ] ...

## AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

Bitmap graphics based programs that are not SunView based can be run from SunView using **overview**. **overview** shows an icon in SunView when **overview** is brought up iconic (**–Wi** flag) or when the program being run by **overview** is suspended (for example, using CTRL-Z). Opening the **overview** icon, or starting **overview** non-iconic, starts the program named on the command line. **overview** suppresses SunView so that SunView window applications will not interfere with the program's display output or input devices.

**overview** runs programs that fit the following profile:

| | |
|---|---|
| *own display* | The program needs to own the bits on the screen. It does not use the *sunview* library to arbitrate the use of the display and input devices between processes. |
| *keyboard input from* **stdin** | The program takes keyboard input from *stdin* directly. |
| *mouse input from* **/dev/mouse** | The program takes locator input from the mouse directly. |

## OPTIONS

**–w** This flag is used to specify that the program being run creates its own SunWindows window in order to receive the serialized input stream from the keyboard and mouse that is provided by the SunWindows kernel driver. **–w** tells **overview** to not convert SunWindows input into ASCII which is then sent to the program being run under **overview** via a pty. *X* and *NeWS* are programs that fall in this category (as of Dec 86, which is subject to change in the future).

## SEE ALSO

**sunview(1)**

## BUGS

Users of **overview** on a Sun-3/110 should be aware of the impact of plane groups on pre-3.2 applications. You cannot successfully run pre-3.2 applications under **overview** if **overview** itself is running in the color buffer. If you start **overview** so that it is not running in the overlay plane, then the enable plane is not be properly set up for viewing the application. This means that you cannot run **overview** with the **-Wf** or **-Wb** generic tool arguments. Also, you cannot run **overview** on a desktop created by **sunview** using the **-8bit_color_only** option.

## NAME

pack, pcat, unpack – compress and expand files

## SYNOPSIS

**pack** [ – ] [ **–f** ] *filename*...

**pcat** *filename*...

**unpack** *filename*...

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**pack** attempts to store the specified files in a packed form using Huffman (minimum redundancy) codes on a byte-by-byte basis. Wherever possible (and useful), each input file *filename* is replaced by a packed file *filename*.**z** with the same access modes, access and modified dates, and owner as those of *filename*. If **pack** is successful, *filename* will be removed.

Packed files can be restored to their original form using **unpack** or **pcat**.

The amount of compression obtained depends on the size of the input file and the frequency distribution of its characters.

Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks unless the distribution of characters is very skewed. This may occur with printer plots or pictures.

Typically, large text-files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression. Their packed versions come in at about 90% of the original size.

No packing will occur if:

the file appears to be already packed
the file name has more than 12 characters
the file has links
the file is a directory
the file cannot be opened
no disk storage blocks will be saved by packing
a file called *name*.**z** already exists
the .**z** file cannot be created
an I/O error occurred during processing

The last segment of the filename must contain no more than 12 characters to allow space for the appended .**z** extension. Directories cannot be packed.

**pcat** does for packed files what **cat**(1V) does for ordinary files, except that **pcat** cannot be used as a filter. The specified files are unpacked and written to the standard output. To view a packed file named *name*.**z** use:

**pcat** *filename*.**z**

or just:

**pcat** *filename*

To make an unpacked copy without destroying the packed version, use

**pcat** *filename* > *newname*

Failure may occur if:

the filename (exclusive of the **.z**) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of **pack**.

**unpack** expands files created by **pack**. For each file *filename* specified in the command, a search is made for a file called *filename*.**z** (or just *filename*, if *filename* ends in **.z**). If this file appears to be a packed, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file. Failure may occur for the same reasons that it may in **pcat**, as well as for the following:

- a file with the "unpacked" name already exists
- the unpacked file cannot be created.

OPTIONS

**–** Print compression statistics for the following filename or names on the standard output. Subsequent '–'s between filenames toggle statistics off and on.

**–f** Force packing of *filename*. This is useful for causing an entire directory to be packed, even if some of the files will not benefit.

DIAGNOSTICS

**pack** returns the number of files that it failed to compress.

**pcat** returns the number of files it was unable to unpack.

**unpack** returns the number of files it was unable to unpack.

SEE ALSO

**cat**(1V)

**NAME**

pagesize – display the size of a page of memory

**SYNOPSIS**

**pagesize**

**DESCRIPTION**

**pagesize** prints the size of a page of memory in bytes, as returned by **getpagesize(2)**. This program is useful in constructing portable shell scripts.

**SEE ALSO**

**getpagesize(2)**

## NAME

passwd, chfn, chsh – change password file information

## SYNOPSIS

**passwd** [ **–fs** ] [ **–F** *filename* ] [ *username* ]

**chfn** [ **–f** ] [ **–F** *filename* ] [ *username* ]

**chsh** [ **–s** ] [ **–F** *filename* ] [ *username* ]

## DESCRIPTION

**passwd** changes (or installs) a password, login shell (**–s** option), or full name (**–f** option) associated with the user *username* (your own by default). **chsh** is equivalent to **passwd** with the **–s** option, and **chfn** is equivalent to **passwd** with the **–f** option.

When changing a password, **passwd** prompts for the old password and then for the new one. You must supply both, and the new password must be typed twice to forestall mistakes.

New passwords should be at least five characters long, if they combine upper-case and lower-case letters, or at least six characters long if in monocase. Users that persist in entering shorter passwords are compromising their own security. The number of significant characters in a password is eight, although longer passwords will be accepted.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

When changing a login shell, **passwd** displays the current login shell and then prompts for the new one. The new login shell must be one of the approved shells listed in **/etc/shells** unless you are the super-user. If **/etc/shells** does not exist, the only shells that may be specified are **/bin/sh** and **/bin/csh**.

The super-user may change anyone's login shell; normal users may only change their own login shell.

When changing a full name, **passwd** displays the current full name, enclosed between brackets, and prompts for a new full name. If you type a carriage return, the full name is not changed. If the full name is to be made blank, you must type the word "none".

The super-user may change anyone's full name; normal users may only change their own.

Use **yppasswd**(1) to change your password in the network Yellow Pages. This will not affect your local password, or your password on any remote machines on which you have accounts. On Sun386i systems, **passwd** calls **yppasswd** automatically if you do not have an entry in the local passwd file.

## OPTIONS

**–s** Change the login shell.

**–f** Change the full name.

**–F** *filename*
Treat *filename* as the password file.

## FILES

| | |
|---|---|
| **/etc/passwd** | file containing all of this information |
| **/etc/shells** | The list of approved shells |

## SEE ALSO

**finger**(1), **login**(1), **yppasswd**(1), **crypt**(3), **passwd**(5),

## BUGS

**passwd** changes a local password, but not a password in the network Yellow Pages. Refer to **yppasswd**(1) for information on how to change a YP password.

There is no way to change the login shell or the full name in the Yellow Pages.

## NAME

paste – join corresponding lines of several files, or subsequent lines of one file

## SYNOPSIS

**paste** *filename1 filename2 ...*
**paste** **–d***list filename1 filename2 ...*
**paste** **–s** [ **–d***list* ] *filename1 filename2 ...*

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

In the first two forms, **paste** concatenates corresponding lines of the given input files *filename1*, *filename2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of **cat**(1V) which concatenates vertically, that is, one file after the other. In the last form above, **paste** replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the TAB character, or with characters from an optionally specified *list*. **paste** can be used as a filter, if – is used in place of a filename.

## OPTIONS

**–d** Without this option, the NEWLINE characters of each but the last file (or last line in case of the **–s** option) are replaced by a TAB character. This option allows replacing the TAB character by one or more alternate characters in *list*. The list is used circularly; when exhausted, it is reused. In parallel merging (no **–s** option), the lines from the last file are always terminated with a NEWLINE character, not from the *list*. *list* may contain the special escape sequences: **\n (NEWLINE), \t (tab), \\ (backslash),**and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell.

**–s** Merge subsequent lines rather than one from each input file. Use TAB for concatenation, unless *list* is specified with **–d**. Regardless of the *list*, the very last character of the file is forced to be a NEWLINE.

## EXAMPLES

**ls | paste – – – –**
List directory in four columns.

**paste –s –d"\t\n"** *filename*
Combine pairs of lines into lines.

## SEE ALSO

**cat**(1V), **cut**(1), **grep**(1V), **pr**(1V)

## DIAGNOSTICS

**line too long**
Output lines are restricted to 511 characters.

**too many files**
Except for **–s** option, no more than 12 input files may be specified.

## NAME

perfmeter – display system performance values in a meter or strip chart

## SYNOPSIS

**perfmeter** [ **–s** *sample-time* ] [ **–h** *h-hand-int* ] [ **–m** *m-hand-int* ]
[ **–M** *smax minmax maxmax* ] [ **–v** *value* ] [ *hostname* ]

## AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**perfmeter** is a SunView utility that displays performance values (statistics) for a given *hostname*. If no host is specified, statistics the current host is metered. The **rstatd**(8C) daemon must be running on the machine being metered.

When open, **perfmeter** displays a performance value in the form of a strip chart. When closed, it displays a meter dial. By default, the display is updated with a *sample-time* of 2 seconds. The hour hand of the meter represents the average over a 20-second interval; the minute hand, the average over 2 seconds. The default value displayed is the percent of CPU being utilized.

The maximum scale value for the strip chart will automatically double or halve to accommodate increasing or decreasing values for the host machine. This scale can be restricted to a certain range with the **–M** option.

## OPTIONS

**–s** *sample-time*
Set the sample time to *sample-time* seconds.

**–h** *h-hand-int*
Set the hour-hand interval to *h-hand-int* seconds.

**–m** *m-hand-int*
Set the minute hand interval to *m-hand-int seconds.*

**–M** *smax minmax maxmax*
Set a range of maximum values for the strip chart. Values for each of the arguments should be powers of 2. *smax* sets the starting maximum-value. *minmax* sets the lowest allowed maximum-value for the scale. *maxmax* sets the highest allowed maximum-value.

**–v** *value*
Set the performance value to be monitored to one of:

| | |
|---|---|
| **cpu** | Percent of CPU being utilized. |
| **pkts** | Ethernet packets per second. |
| **page** | Paging activity in pages per second. |
| **swap** | Jobs swapped per second. |
| **intr** | Number of device interrupts per second. |
| **disk** | Disk traffic in transfers per second. |
| **cntxt** | Number of context switches per second. |
| **load** | Average number of runnable processes over the last minute. |
| **colls** | Collisions per second detected on the ethernet. |
| **errs** | Errors per second on receiving packets. |

## USAGE

### Commands

You can change the statistic being displayed by clicking the RIGHT mouse button, and then choosing the desired menu item. The other meter parameters can be modified by moving the pointer onto the tool (either open or closed), and typing:

| | |
|---|---|
| **m** | Decrease *minutehandintv* by one |
| **M** | Increase *minutehandintv* by one |
| **h** | Decrease *hourhandintv* by one |
| **H** | Increase *hourhandintv* by one |
| **1-9** | Set *sampletime* to a range from 1 to 9 seconds. |

## FILES

| | |
|---|---|
| **/etc/servers** | starts statistics server |

## SEE ALSO

**sunview**(1), **netstat**(8C), **rstatd**(8C), **vmstat**(8)

**NAME**

pg – page through a file on a soft-copy terminal

**SYNOPSIS**

**/usr/5bin/pg** [ **–cefns** ] [ *–number* ] [ **–p string** ] [ *+linenumber* ] [ *+/pattern/* ] [ **filename** . . . ]

**DESCRIPTION**

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command.

The **pg** command is a filter that allows you to page through *filename*, one screenful at a time, on a soft-copy terminal. With a *filename* of '–', or no *filename* specified, **pg** reads from the standard input. Each screenful is followed by a prompt. If the user types a RETURN, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, **pg** scans the **terminfo**(5V) data base for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal type **dumb** is assumed.

The responses that may be typed when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)< NEWLINE > or < SPACE >
: Display one page. The address is specified in pages.

(+1) **l** With a relative address **pg** will simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
: Simulate scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Redisplay the current page of text.

**$** Display the last full window in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in **ed**(1) are available. They must always be terminated by a < NEWLINE >, even if the **–n** option is specified.

*i/pattern/*
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
*i?pattern?*
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the **?**.

After searching, **pg** will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of **pg** can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

**s** *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a < NEWLINE >, even if the **–n** option is specified.

**h** Help by displaying an abbreviated summary of available commands.

**q** or **Q** Quit **pg**.

**!***command*
**command** is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a < NEWLINE >, even if the **–n** option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key, normally **^(CTRL backslash) or the** BREAK (interrupt) key. This causes **pg** to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then **pg** acts just like **cat**(1V), except that a header is printed before each file (if there is more than one).

OPTIONS

The command line options are:

**–***number*
An integer specifying the size (in lines) of the window that **pg** is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

**–p** *string*
Use **string** as the prompt. If the prompt string contains a '**%d**', the first occurrence of '**%d**' in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is '**:**'.

**–c** Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the **terminfo**(5V) data base.

**–e** do *not* pause at the end of each file.

**–f** Inhibit **pg** from splitting lines. Normally, **pg** splits lines longer than the screen width, but some sequences of characters in the text being displayed (for instance, escape sequences for underlining) generate undesirable results. The

**–n** Automatic end of command as soon as a command letter is entered. Normally, commands must be terminated by a < NEWLINE > character.

**–s** Print all messages and prompts in standout mode (usually inverse video).

**+***linenumber*
Start up at *linenumber*.

*+/pattern/*
Start up at the first line containing the regular expression pattern.

**EXAMPLE**

A sample usage of **pg** in reading system news would be

**news | pg –p "(Page %d):"**

**FILES**

| | |
|---|---|
| **/usr/share/lib/terminfo/*** | terminal information data base |
| **/tmp/pg*** | temporary file when input is from a pipe |

**SEE ALSO**

**cat**(1V), **crypt**(1), **ed**(1), **grep**(1V), **more**(1), **terminfo**(5V)

**BUGS**

If terminal TAB characters are not set every eight positions, undesirable results may occur.

When using **pg** as a filter with another command that changes the terminal I/O options (for instance, **crypt**(1)), terminal settings may not be restored correctly.

**NOTES**

While waiting for terminal input, **pg** responds to BREAK , DEL , and ^ by terminating execution. Between prompts, however, these signals interrupt **pg**'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of **more**(1) will find that the z and f commands are available, and that the terminal /,^,**or**? may be omitted from the searching commands.

NAME

plot, aedplot, bgplot, crtplot, dumbplot, gigiplot, hpplot, implot, t300, t300s, t4013, t450, tek – graphics filters for various plotters

SYNOPSIS

**plot** [ –T*terminal* ]

DESCRIPTION

**plot** reads plotting instructions (see **plot**(5)) from the standard input and produces plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* is specified, the environment variable **TERM** is used. The default *terminal* is **tek**.

ENVIRONMENT

Except for **ver**, the following terminal-types can be used with '**lpr –g**' see **lpr**(1)) to produce plotted output:

**2648 | 2648a | h8 | hp2648 | hp2648a**
Hewlett Packard 2648 graphics terminal.

**300** DASI 300 or GSI terminal (Diablo mechanism).

**300s | 300S** DASI 300s terminal (Diablo mechanism).

**450** DASI Hyterm 450 terminal (Diablo mechanism).

**4013** Tektronix 4013 storage scope.

**4014 | tek** Tektronix 4014 and 4015 storage scope with Enhanced Graphics Module. (Use 4013 for Tektronix 4014 or 4015 without the Enhanced Graphics Module).

**aed** AED 512 color graphics terminal.

**bgplot | bitgraph**
BBN bitgraph graphics terminal.

**crt** Any crt terminal capable of running **vi**(1).

**dumb | un | unknown**
Dumb terminals without cursor addressing or line printers.

**gigi | vt125** DEC vt125 terminal.

**h7 | hp7 | hp7221**
Hewlett Packard 7221 graphics terminal.

**implot** Imagen plotter.

**var** Benson Varian printer-plotter

**ver** Versatec D1200A printer-plotter. The output is scan-converted and suitable input to '**lpr –v**'.

FILES

**/usr/bin/t300**
**/usr/bin/t300s**
**/usr/bin/t4013**
**/usr/bin/t450**
**/usr/bin/aedplot**
**/usr/bin/crtplot**
**/usr/bin/gigiplot**
**/usr/bin/implot**
**/usr/bin/plot**
**/usr/bin/bgplot**
**/usr/bin/dumbplot**
**/usr/bin/hpplot**

**/usr/bin/vplot**
**/usr/bin/tek**
**/usr/bin/t450**
**/usr/bin/t300**
**/usr/bin/t300s**
**/usr/bin/vplot**
**/var/tmp/vplot***nnnnnn*

SEE ALSO

**lpr**(1), **vi**(1), **graph**(1G), **plot**(3X), **plot**(5), **rasterfile**(5)

## NAME

pr – prepare file(s) for printing, perhaps in multiple columns

## SYNOPSIS

**pr** [ −|+ *n* ] [ **−fmt** ] [ **−h** *string* ] [ **−l***n* ] [ **−s***c* ] [ **−w***n* ] [ *filename* ] ...

## SYSTEM V SYNOPSIS

**/usr/5bin/pr** [ −|+ *n* ] [ **−adfmprt** ] [ **−e***ck* ] [ **−h** *string* ]
[ **−i***ck* ] [ **−l***n* ] [ **−n***ck* ] [ **−o***n* ] [ **−s***c* ]
[ **−w***n* ] [ *filename* ] ...

## DESCRIPTION

**pr** prepares one or more *filename*s for printing. By default, the output is separated into pages headed by a date, the name of the file, and the page number. **pr** prints its standard input if there are no *filename* arguments. FORMFEED characters in the input files cause page breaks in the output, as expected.

By default, columns are of equal width, separated by at least one SPACE; lines that do not fit are truncated. If the **−s** option is used, lines are not truncated and columns are separated by the separation character.

Inter-terminal messages using **write**(1) are forbidden during a **pr**.

## OPTIONS

Options apply to all following *filename*s but may be reset between *filename*s:

**−f** Use FORMFEED characters instead of NEWLINE characters to separate pages. A FORMFEED 1is assumed to use up two blank lines at the top of a page. Thus this option does not affect the effective page length.

**−m** Print all *filename*s simultaneously, each in one column, for example:

| | | |
|---|---|---|
| **Print** | **Print** | **The** |
| **the** | **the** | **third** |
| **lines** | **lines** | **file's** |
| **of** | **of** | **lines** |
| **file** | **file** | **go** |
| **one.** | **two.** | **here.** |

**−t** Do not print the 5-line header or the 5-line trailer normally supplied for each page. Pages are not separated when this option is used, even if the **−f** option was used. The **−t** option is intended for applications where the results should be directed to a file for further processing.

**−h** *string*
Use *string*, instead of the file name, in the page header.

**−l***n* Take the length of the page to be *n* lines instead of the default 66.

**−s***c* Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a TAB.

**−w***n* For multicolumn output, take the width of the page to be *n* characters instead of the default 72.

**−***n* Produce *n*-column output. For example:

| | | |
|---|---|---|
| **Print** | **of** | **in** |
| **the** | **one** | **three** |
| **lines** | **file** | **columns.** |

Columns are not balanced; if, for example, there are as many lines in the file as there are lines on the page, only one column will be printed. Even if the **−t** option (see below) is specified, blank lines will be printed at the end of the output to pad it to a full page.

**+***n* Begin printing with page *n*.

SYSTEM V OPTIONS

When the *–n* option is specified for multicolumn output, columns are balanced. For example, if there are as many lines in the file as there are lines to be printed, and two columns are to be printed, each column will contain half the lines of the file. If the **–t** option is specified, no blank lines will be printed to pad the last page.

The options **–e** and **–i** are assumed for multicolumn output. The **–m** option overrides the **–k** and **–a** options.

The **–f** option does not assume that FORMFEED uses up two blank lines; blank lines will be printed after the FORMFEED if necessary.

**–a** When combined with the *–n* option, print multicolumn output across the page. For example:

**Print the lines**
**of one file**
**in three columns.**

**–d** Double-space the output.

**–p** Pause before beginning each page if the output is directed to a terminal (*pr*) will ring the bell at the terminal and wait for a RETURN).

**–r** Do Not print diagnostic reports if a file cannot be opened, or if it is empty.

**–e***ck* Expand *input* TAB characters to character positions $k+1$, $2*k+1$, $3*k+1$, etc. If $k$ is 0 or is omitted, default TAB settings at every eighth position are assumed. TAB characters in the input are expanded into the appropriate number of SPACE characters. If *c* (any non-digit character) is given, it is treated as the input TAB character (default for *c* is the TAB character).

**–i***ck* In *output*, replace white space wherever possible by inserting TAB characters to character positions $k+1$, $2*k+1$, $3*k+1$, etc. If $k$ is 0 or is omitted, default TAB settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output TAB (default for *c* is the TAB character).

**–n***ck* Provide *k*-digit line numbering (default for $k$ is 5). The number occupies the first $k+1$ character positions of each column of normal output or each line of **–m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a TAB).

**–o***k* Offset each line by *k* character positions. The number of character positions per line is the sum of the width and offset.

EXAMPLES

Print a file called **dreadnought** on the printer — this is the simplest use of **pr**:

```
example% pr dreadnought | lpr
example%
```

Produce three laminations of a file called **ridings** side by side in the output, with no headers or trailers, the results to appear in the file called **Yorkshire**:

```
example% pr –m –t ridings ridings
example%
```

FILES

**/dev/tty*** to suspend messages.

SEE ALSO

**lpr**(1), **write**(1), **cat**(1V)

DIAGNOSTICS

**can't print 0 cols, using 1**
–**0** was specified as a –*n* option.

**pr: bad key**
*key* An illegal option was given.

**pr: No room for columns.**
The number of columns requested won't fit on the page.

**pr: Too many args**
More than 10 files were specified with the –**m** option.

**file:** *error*
**filename** could not be opened. This diagnostic is not printed if **pr** is printing on a terminal.

SYSTEM V DIAGNOSTICS

**pr: bad option**
An illegal option was given.

**pr: width too small**
The number of columns requested won't fit on the page.

**pr: too many files**
More than 10 files were specified with the –**m** option.

**pr: page-buffer overflow**
The formatting required is more complicated than **pr** can handle.

**pr: out of space**
**pr** could not allocate a buffer it required.

**pr:***filename* **-- empty file**
**filename** was empty. This diagnostic is printed after all the files are printed if **pr** is printing on a terminal.

**pr: can't open** *filename*
**filename** could not be opened. This diagnostic is printed after all the files are printed if **pr** is printing on a terminal.

BUGS

The options described above interact with each other in strange and as yet to be defined ways.

NAME

printenv – display environment variables currently set

SYNOPSIS

**printenv** [ *variable* ]

DESCRIPTION

**printenv** prints out the values of the variables in the environment. If a *variable* is specified, only its value is printed.

SEE ALSO

**csh**(1), **sh**(1), **stty**(1V), **tset**(1), **environ**(5V)

DIAGNOSTICS

If a *variable* is specified and it is not defined in the environment, **printenv** returns an exit status of **1.**

## NAME

prof – display profile data

## SYNOPSIS

**prof** [ **–alnsz** ] [ **–v** *–low* [ *–high* ] ] [ *image-file* [ *profile-file* ... ] ]

## DESCRIPTION

**prof** produces an execution profile of a program. The profile data is taken from the profile file which is created by programs compiled with the **–p** option of **cc**(1V) and other compilers. That option also links in versions of the library routines (see **monitor**(3)) which are compiled for profiling. The symbol table in the executable image file *image-file* (**a.out** by default) is read and correlated with the profile file *profile-file* (**mon.out** by default). For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call. If more than one profile file is specified, the **prof** output shows the sum of the profile information in the given profile files.

To tally the number of calls to a routine, the modules that make up the program must be compiled with the '**cc –p**' option (see **cc**(1v)). This option also means that the profile file is produced automatically.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(3)).

Beware of quantization errors.

The profiled program must call **exit**(2) or return normally for the profiling information to be saved in the **mon.out** file.

## OPTIONS

**–a** Report all symbols rather than just external symbols.

**–l** Sort the output by symbol value.

**–n** Sort the output by number of calls.

**–s** Produce a summary profile file in **mon.sum**. This is really only useful when more than one profile file is specified.

**–z** Display routines which have zero usage (as indicated by call counts and accumulated time).

**–v** [ *–low* [ *–high* ]]
Suppress all printing and produce a graphic version of the profile on the standard output for display by the **plot**(1G) filters. When plotting, the numbers *low* and *high*, (by default 0 and 100), select a percentage of the profile to be plotted, with accordingly higher resolution.

## ENVIRONMENT

**PROFDIR**
If this environment variable contains a value, place profiling output within that directory, in a file named *pid.programname*. *pid* is the process ID, and *programname* is the name of the program being profiled, as determined by removing any path prefix from the **argv[0]** with which the program was called. If the variable contains a NULL value, no profiling output is produced. Otherwise, profiling output is placed in the file **mon.out**.

## FILES

| | |
|---|---|
| **a.out** | executable file containing namelist |
| **$PROFDIR/***pid.programname* | |
| **mon.out** | profiling output |
| **mon.sum** | summary profile |

## SEE ALSO

**gprof**(1), **tcov**(1), **plot**(1G), **cc**(1V), **exit**(2), **profil**(2), **monitor**(3)

BUGS

**prof** is confused by the FORTRAN compiler which puts the entry points at the bottom of subroutines and functions.

## NAME

prs – display selected portions an SCCS history

## SYNOPSIS

**/usr/sccs/prs** [ **–ael** ] [ **–d** [ *dataspec* ] ] [ **–r** [ *SID* ] ] *filename* ...

## DESCRIPTION

**prs** prints, on the standard output, parts or all of an SCCS file (see **sccsfile(5)**) in a user supplied format. If a directory is named, **prs** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of '–' is given, the standard input is read, in which case each line is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

## OPTIONS

Options apply independently to each named file.

**–a** Request printing of information for both removed, that is, delta type = *R*, (see **rmdel**(1)) and existing, that is, delta type = *D*, deltas. If the **–a** option is not specified, information for existing deltas only is provided.

**–e** Request information for all deltas created *earlier* than and including the delta designated using the **–r** option.

**–l** Request information for all deltas created *later* than and including the delta designated using the **–r** option.

**–d** [ *dataspec* ]
Specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see **DATA KEYWORDS**) interspersed with optional user supplied text.

**–r** [ *SID* ]
Specify the SCCS Identification SID string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

In the absence of the **–d** options, **prs** displays a default set of information consisting of: delta-type, release number and level number, date and time last changed, user-name of the person who changed the file, lines inserted, changed, and unchanged, the MR numbers, and the comments.

## DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see **sccsfile**(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by **prs** consists of: 1) the user supplied text; and 2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a RETURN.

User supplied text is any text other than recognized data keywords. A TAB is specified by **\t** and RETURN/NEWLINE is specified by **\n**.

TABLE 1. SCCS Files Data Keywords

| *Keyword* | *Data Item* | *File Section* | *Value* | *Format* |
|---|---|---|---|---|
| :Dt: | Delta information | Delta Table | See below* | S |
| :DL: | Delta line statistics | " | :Li:/:Ld:/:Lu: | S |
| :Li: | Lines inserted by Delta | " | nnnnn | S |
| :Ld: | Lines deleted by Delta | " | nnnnn | S |
| :Lu: | Lines unchanged by Delta | " | nnnnn | S |
| :DT: | Delta type | " | *D* or *R* | S |
| :I: | SCCS ID string (SID) | " | :R:.:L:.:B:.:S: | S |
| :R: | Release number | " | nnnn | S |
| :L: | Level number | " | nnnn | S |
| :B: | Branch number | " | nnnn | S |
| :S: | Sequence number | " | nnnn | S |
| :D: | Date Delta created | " | :Dy:/:Dm:/:Dd: | S |
| :Dy: | Year Delta created | " | nn | S |
| :Dm: | Month Delta created | " | nn | S |
| :Dd: | Day Delta created | " | nn | S |
| :T: | Time Delta created | " | :Th::Tm::Ts: | S |
| :Th: | Hour Delta created | " | nn | S |
| :Tm: | Minutes Delta created | " | nn | S |
| :Ts: | Seconds Delta created | " | nn | S |
| :P: | Programmer who created Delta | " | logname | S |
| :DS: | Delta sequence number | " | nnnn | S |
| :DP: | Predecessor Delta seq-no. | " | nnnn | S |
| :DI: | Seq-no. of deltas incl., excl., ignored | " | :Dn:/:Dx:/:Dg: | S |
| :Dn: | Deltas included (seq #) | " | :DS: :DS: ... | S |
| :Dx: | Deltas excluded (seq #) | " | :DS: :DS: ... | S |
| :Dg: | Deltas ignored (seq #) | " | :DS: :DS: ... | S |
| :MR: | MR numbers for delta | " | text | M |
| :C: | Comments for delta | " | text | M |
| :UN: | User names | User Names | text | M |
| :FL: | Flag list | Flags | text | M |
| :Y: | Module type flag | " | text | S |
| :MF: | MR validation flag | " | *yes* or *no* | S |
| :MP: | MR validation pgm name | " | text | S |
| :KF: | Keyword error/warning flag | " | *yes* or *no* | S |
| :BF: | Branch flag | " | *yes* or *no* | S |
| :J: | Joint edit flag | " | *yes* or *no* | S |
| :LK: | Locked releases | " | :R: ... | S |
| :Q: | User defined keyword | " | text | S |
| :M: | Module name | " | text | S |
| :FB: | Floor boundary | " | :R: | S |
| :CB: | Ceiling boundary | " | :R: | S |
| :Ds: | Default SID | " | :I: | S |
| :ND: | Null delta flag | " | *yes* or *no* | S |
| :FD: | File descriptive text | Comments | text | M |
| :BD: | Body | Body | text | M |
| :GB: | Gotten body | " | text | M |
| :W: | A form of **what** string | N/A | :Z::M:\t:I: | S |
| :A: | A form of **what** string | N/A | :Z::Y: :M: :I::Z: | S |
| :Z: | **what** string delimiter | N/A | @(#) | S |
| :F: | SCCS file name | N/A | text | S |
| :PN: | SCCS file path name | N/A | text | S |

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

The following command:

**/usr/sccs/prs –d"Users and/or user IDs for :F: are:\n:UN:" s.file**

may produce on the standard output:

**Users and/or user IDs for s.file are:**
**xyz**
**131**
**abc**

The command:

**/usr/sccs/prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file**

may produce on the standard output:

**Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas**

As a *special case*:

**/usr/sccs/prs s.file**

may produce on the standard output:

**D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000**
**MRs:**
**bl78-12345**
**bl79-54321**
**COMMENTS:**
**this is the comment line for s.file initial delta**

for each delta table entry of the "D" type. The only option argument allowed to be used with the *special case* is the **–a** option.

FILES

**/tmp/pr?????**

SEE ALSO

**admin**(1), **delta**(1), **get**(1), **help**(1), **sccs**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

DIAGNOSTICS

Use **help**(1) for explanations.

NAME

prt – display the delta and commentary history of an SCCS file

SYNOPSIS

**/usr/sccs/prt** [**–abdefistu**] [**–y**[*SID* ]] [**–c**[*cutoff*]] [**–r**[*rev-cutoff*] ] *filename...*

DESCRIPTION

Note: the **prt** command is an older version of **prs**(1) that in most circumstances is more convenient to use, but is less flexible than **prs**.

**prt** prints part or all of an SCCS file in a useful format. If a directory is named, **prt** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files are silently ignored. If a name of '–' is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

**–a** Print those types of deltas normally not printed by the **d** keyletter. These are types **R** (removed). This keyletter is effective only if the **d** keyletter is also specified (or assumed).

**–b** Print the body of the SCCS file.

**–d** This keyletter normally prints delta table entries of the **D** type.

**–e** This keyletter implies the **d**, **i**, **u**, **f**, and **t** keyletters and is provided for convenience.

**–f** Print the flags of the named file.

**–i** Print the serial numbers of those deltas included, excluded, and ignored. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

The following format is used to print those portions of the SCCS file as specified by the above keyletters. The printing of each delta table entry is preceded by a NEWLINE.

- Type of delta (**D** or **R**).
- SPACE.
- SCCS identification string (SID).
- TAB.
- Date and time of creation (in the form YY/MM/DD HH:MM:SS).
- SPACE.
- Creator.
- TAB.
- Serial number.
- SPACE.
- Predecessor delta's serial number.
- TAB.
- Statistics (in the form inserted/deleted/unchanged).
- NEWLINE.
- "Included:TAB", followed by SID's of deltas included, followed by NEWLINE (only if there were any such deltas and if **i** keyletter was supplied).
- "Excluded:TAB", followed by SID's of deltas excluded, followed by NEWLINE (see note above).
- "Ignored:TAB", followed by SID's of deltas ignored, followed by NEWLINE (see note above).
- "MRs:TAB", followed by MR numbers related to the delta, followed by NEWLINE (only if any MR numbers were supplied).
- Lines of comments (delta commentary), followed by newline (if any were supplied).

**–s** Print only the first line of the delta table entries; that is, only up to the statistics. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

**-t** Print the descriptive text contained in the file.

**-u** Print the login-names and/or numerical group IDs of those users allowed to make deltas.

**-y**[*SID*] Print the delta table entries to stop when the delta just printed has the specified SID. If no delta in the table has the specified SID, the entire table is printed. If no SID is specified, the first delta in the delta table is printed. This keyletter will print the entire delta table entry for each delta as a single line (the NEWLINE in the normal multi-line format of the **d** keyletter are replaced by SPACE characters) preceded by the name of the SCCS file being processed, followed by a :, followed by a TAB. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

**-c**[*cutoff*]
Stop printing the delta table entries if the delta about to be printed is older than the specified cutoff date-time (see **get**(1) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

**-r**[*rev-cutoff*]
Begin printing the delta table entries when the delta about to be printed is older than or equal to the specified cutoff date-time (see **get**(1) for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. (In this case, nothing will be printed). As with the **y** keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a :, followed by a tab. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

If any keyletter but **y**, **c**, or **r** is supplied, the name of the file being processed (preceded by one NEWLINE and followed by two NEWLINE characters) is printed before its contents.

If none of the **u**, **f**, **t**, or **b** keyletters is supplied, the **d** keyletter is assumed.

Note: the **s** and **i** keyletters, and the **c** and **r** keyletters are mutually exclusive; therefore, they may not be specified together on the same **prt** command.

The form of the delta table as produced by the **y**, **c**, and **r** keyletters makes it easy to sort multiple delta tables in chronological order.

When both the **y** and **c** or the **y** and **r** keyletters are supplied, **prt** will stop printing when the first of the two conditions is met.

## SEE ALSO

**admin**(1), **get**(1), **delta**(1), **prs**(1), **what**(1), **help**(1), **sccs**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use **help**(1) for explanations.

NAME

ps – display the status of current processes

SYNOPSIS

**ps** [ **–acCegklnrStuvwxU** ] [ *num* ] [ *kernel_name* ] [ *c_dump_file* ] [ *swap_file* ]

DESCRIPTION

**ps** displays information about processes. Normally, only those processes that are running with your effective user ID and are attached to a controlling terminal (see **termio**(4)) are shown. Additional categories of processes can be added to the display using various options. In particular, the **–a** option allows you to include processes that are not owned by you (that do not have your user ID), and the **–x** option allows you to include processes without control terminals. When you specify both **–a** and **–x**, you get processes owned by anyone, with or without a control terminal. The **–r** option restricts the list of processes printed to "running" processes: runnable processes, those in page wait, or those in disk or other short-term waits.

**ps** displays the process ID, under PID; the control terminal (if any), under TT; the cpu time used by the process so far, including both user and system time), under CPU; the state of the process, under STAT; and finally, an indication of the COMMAND that is running.

The state is given by a sequence of four letters, for example, '**RWNA**'.

*First letter* indicates the runnability of the process:

- **R** Runnable processes,
- **T** Stopped processes,
- **P** Processes in page wait,
- **D** Processes in disk (or other short term) waits,
- **S** Processes sleeping for less than about 20 seconds,
- **I** Processes that are idle (sleeping longer than about 20 seconds),
- **Z** Processes that have terminated and that are waiting for their parent process to do a **wait**(2) ("zombie" processes).

*Second letter* indicates whether a process is swapped out;

- *blank* (that is, a SPACE) in this position indicates that the process is loaded (in memory).
- **W** Process is swapped out.
- **>** Process has specified a soft limit on memory requirements and has exceeded that limit; such a process is (necessarily) not swapped.

*Third letter* indicates whether a process is running with altered CPU scheduling priority (nice):

- *blank* (that is, a SPACE) in this position indicates that the process is running without special treatment.
- **N** The process priority is reduced,
- **<** The process priority has been raised artificially.

*Fourth letter* indicates any special treatment of the process for virtual memory replacement. The letters correspond to options to the **vadvise**(2) system call. Currently the possibilities are:

- *blank* (that is, a SPACE) in this position stands for **VA_NORM**.
- **A** Stands for **VA_ANOM**. An **A** typically represents a program which is doing garbage collection.
- **S** Stands for **VA_SEQL**. An **S** is typical of large image processing programs that are using virtual memory to sequentially address voluminous data.

*kernel_name* specifies the location of the system namelist. If the **–k** option is given, *c_dump_file* tells **ps** where to look for the core dump. Otherwise, the core dump is located in the file **/vmcore** and this argument is ignored. *swap_file* gives the location of a swap file other than the default, **/dev/drum**.

OPTIONS

**–a** Include information about processes owned by others.

**-c** Display the command name, as stored internally in the system for purposes of accounting, rather than the command arguments, which are kept in the process' address space. This is more reliable, if less informative, since the process is free to destroy the latter information.

**-C** Display raw CPU time in the %CPU field instead of the decaying average.

**-e** Display the environment as well as the arguments to the command.

**-g** Display all processes. Without this option, **ps** only prints "interesting" processes. Processes are deemed to be uninteresting if they are process group leaders. This normally eliminates top-level command interpreters and processes waiting for users to login on free terminals.

**-k** Normally, *kernel_name* defaults to **/vmunix**, *c_dump_file* is ignored, and *swap_file* defaults to **/dev/drum**. With the **-k** option in effect, these arguments default to **/vmunix**, **/vmcore**, and **/dev/drum**, respectively.

**-l** Display a long listing, with fields **PPID, CP, PRI, NI, SZ, RSS** and **WCHAN** as described below.

**-n** Produce numerical output for some fields. In a long listing, the **WCHAN** field is printed numerically rather than symbolically, or, in a user listing, the **USER** field is replaced by a **UID** field.

**-r** Restrict output to "running" processes.

**-S** Display accumulated CPU time used by this process and all of its reaped children.

**t***x* Restrict output to processes whose controlling terminal is *x* (which should be specified as printed by **ps**, for example, **t3** for **/dev/tty3**, **tco** for **/dev/console**, **td0** for **/dev/ttyd0**, **t?** for processes with no terminal, etc). This option must be the last one given.

**-u** Display user-oriented output. This includes fields **USER, %CPU, %MEM, SZ, RSS** and **START** as described below.

**-v** Display a version of the output containing virtual memory. This includes fields **RE, SL, PAGEIN, SIZE, RSS, LIM, %CPU** and **%MEM**, described below.

**-w** Use a wide output format (132 columns rather than 80); if repeated, that is, **-ww**, use arbitrarily wide output. This information is used to decide how much of long commands to print.

**-x** Include processes with no controlling terminal.

**-U** Update a private database where **ps** keeps system information. Thus, '**ps -U**' should be included in the **/etc/rc** file.

*num* A process number may be given, in which case the output is restricted to that process. This option must also be last.

**DISPLAY FORMATS**

Fields that are not common to all output formats:

**USER** Name of the owner of the process.

**%CPU** CPU utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all **%CPU** fields to exceed 100%.

**NI** Process scheduling increment (see **getpriority**(2) and **nice**(3C).)

**SIZE**

**SZ** The combined size of the data and stack segments (in kilobyte units)

**RSS** Real memory (resident set) size of the process (in kilobyte units).

**LIM** Soft limit on memory used, specified using a call to **getrlimit**(2); if no limit has been specified then shown as *xx*.

**%MEM** Percentage of real memory used by this process.

**RE** Residency time of the process (seconds in core).

**SL** Sleep time of the process (seconds blocked).

**PAGEIN** Number of disk I/O's resulting from references by the process to pages not loaded in core.

**UID** Numerical user-ID of process owner.

**PPID** Numerical ID of parent of process.

**CP** Short-term CPU utilization factor (used in scheduling).

**PRI** Process priority (non-positive when in non-interruptible wait).

**START** Time the process was created if that was today, or the date it was created if that was before today.

**WCHAN** Event on which process is waiting (an address in the system). A symbol is chosen that classifies the address, unless numerical output is requested (see the **n** flag). In this case, the address is printed in hexadecimal.

**F** Flags associated with process as in **<sys/proc.h>**:.

| | | |
|---|---|---|
| **SLOAD** | 0000001 | in core |
| **SSYS** | 0000002 | swapper or pager process |
| **SLOCK** | 0000004 | process being swapped out |
| **SSWAP** | 0000008 | save area flag |
| **STRC** | 0000010 | process is being traced |
| **SWTED** | 0000020 | parent has been told that this process stopped |
| **SULOCK** | 0000040 | user settable lock in core |
| **SPAGE** | 0000080 | process in page wait state |
| **SKEEP** | 0000100 | another flag to prevent swap out |
| **SOMASK** | 0000200 | restore old mask after taking signal |
| **SWEXIT** | 0000400 | working on exiting |
| **SPHYSIO** | 0000800 | doing physical I/O |
| **SVFORK** | 0001000 | process resulted from **vfork**() |
| **SVFDONE** | 0002000 | another vfork flag |
| **SNOVM** | 0004000 | no vm, parent in a **vfork**() |
| **SPAGI** | 0008000 | init data space on demand, from inode |
| **SSEQL** | 0010000 | user warned of sequential vm behavior |
| **SUANOM** | 0020000 | user warned of anomalous vm behavior |
| **STIMO** | 0040000 | timing out during sleep |
| **SPGLDR** | 0080000 | process is session process group leader |
| **STRACNG** | 0100000 | process is tracing another process |
| **SOWEUPC** | 0200000 | owe process an **addupc**() call at next ast |
| **SSEL** | 0400000 | selecting; wakeup/waiting danger |
| **SFAVORD** | 2000000 | favored treatment in swapout and pageout |
| **SLKDONE** | 4000000 | record-locking has been done |
| **STRCSYS** | 8000000 | tracing system calls |

A process that has exited and has a parent, but has not yet been waited for by the parent is marked **<defunct>**; a process that is blocked trying to exit is marked **<exiting>**; otherwise, **ps** makes an educated guess as to the file name and arguments given when the process was created by examining memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

FILES

| | |
|---|---|
| **/vmunix** | system namelist |
| **/dev/kmem** | kernel memory |
| **/dev/drum** | swap device |
| **/vmcore** | core file |
| **/dev** | searched to find swap device and terminal names |
| **/etc/psdatabase** | system namelist, device, and wait channel information |

**SEE ALSO**

**kill**(1), **w**(1), **getpriority**(2), **getrlimit**(2), **wait**(2), **vadvise**(2), **nice**(3C), **termio**(4), **pstat**(8)

**BUGS**

Things can change while **ps** is running; the picture it gives is only a close approximation to the current state.

## NAME

ptx – generate a permuted index

## SYNOPSIS

**ptx** [ **–f** ] [ **–t** ] [ **–w** *n* ] [ **–g** *n* ] [ **–o** *only* ] [ **–i** *ignore* ] [ **–b** *break] [* **–r** ] [ *input* [ *output* ] ]

## AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**ptx** generates a permuted index of the contents of file *input* onto file *output* (defaults are standard input and output). **ptx** has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. **ptx** produces output in the form:

*.xx " tail " " before keyword " " keyword and after " " head "*

where *.xx* may be an **nroff**(1) or **troff**(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

## OPTIONS

**–f** Fold upper and lower case letters for sorting.

**–t** Prepare the output for the phototypesetter; the default line length is 100 characters.

**–w** *n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

**–g** *n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

**–o** *only* Use as keywords only the words given in the *only* file.

**–i** *ignore*
Do not use as keywords any words given in the *ignore* file. If the **–i** and **–o** options are missing, use **/usr/lib/eign** as the *ignore* file.

**–b** *break*
Use the characters in the *break* file to separate words. In any case, TAB, NEWLINE, and SPACE characters are always used as break characters.

**–r** Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

## FILES

**/usr/bin/sort**
**/usr/lib/eign**

## SEE ALSO

**nroff**(1), **troff**(1)

## BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

pwd – display the pathname of the current working directory

SYNOPSIS

**pwd**

DESCRIPTION

**pwd** prints the pathname of the working (current) directory.

If you are using **csh**(1), you can use the **dirs** builtin command to do the same job more quickly; *but* **dirs** can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it. This is because **pwd** searches back up the directory tree to report the true pathname, whereas **dirs** remembers the pathname from the last **cd**(1) command. The example below illustrates the differences.

```
example% cd /usr/wendy/january/reports
example% pwd
/usr/wendy/january/reports
example% dirs
 /january/reports
example% mv /january /february
example% pwd
/usr/wendy/february/reports
example% dirs
 /january/reports
example%
```

**pwd** and **dirs** also give different answers when you change directory through a symbolic link. For example:

```
example% cd /usr/wendy/january/reports
example% pwd
/usr/wendy/january/reports
example% dirs
 /january/reports
example% ls –l /usr/wendy/january
lrwxrwxrwx 1 wendy       17 Jan 30 1983 /usr/wendy/january -> /usr/wendy/1984/jan/
example% cd /usr/wendy/january
example% pwd
/usr/wendy/1984/jan
example% dirs
/usr/wendy/january
```

SEE ALSO

**cd**(1), **csh**(1)

NAME

quota – display a user's disk quota and usage

SYNOPSIS

**quota** [ –v ] [ *username* ]

DESCRIPTION

**quota** displays users' disk usage and limits. Only the super-user may use the optional *username* argument to view the limits of users other than himself.

**quota** without options displays only warnings about mounted file systems where usage is over quota. Remotely mounted file systems which are mounted with the "noquota" option (see **fstab**(5)) are ignored.

OPTIONS

**–v** Display user's quotas on all mounted file systems where quotas exist.

FILES

| | |
|---|---|
| **quotas** | quota file at the file system root |
| **/etc/mtab** | list of currently mounted filesystems |

SEE ALSO

**quotactl**(2), **fstab**(5), **edquota**(8), **quotaon**(8), **rquotad**(8C)

## NAME

ranlib – convert archives to random libraries

## SYNOPSIS

**ranlib** [ **–t** ] *archive...*

## DESCRIPTION

**ranlib** converts each *archive* to a form that can be linked more rapidly. **ranlib** does this by adding a table of contents called **__.SYMDEF** to the beginning of the archive. **ranlib** uses **ar**(1V) to reconstruct the archive. Sufficient temporary file space must be available in the file system that contains the current directory.

## OPTIONS

**–t** **ranlib** only "touches" the archives and does not modify them. This is useful after copying an archive or using the **–t** option of **make**(1) to avoid having **ld**(1) complain about an "out of date" symbol table.

## SEE ALSO

**ar**(1V), **ld**(1), **lorder**(1), **make**(1)

## BUGS

Because generation of a library by **ar** and randomization of the library by **ranlib** are separate processes, phase errors are possible. The linker, **ld**, warns when the modification date of a library is more recent than the creation date of its dictionary; but this means that you get the warning even if you only copy the library.

NAME

rasfilter8to1 – convert an 8-bit deep rasterfile to a 1-bit deep rasterfile

SYNOPSIS

**rasfilter8to1** [ **–d** ] [ **–rgba** *threshold* ] [ *infile* [ *outfile* ]]

DESCRIPTION

**rasfilter8to1** reads the 8-bit deep rasterfile *infile* (the standard input default) and converts it to the 1-bit deep rasterfile *outfile* (standard output default) by thresholding or ordered dither. The output format is Sun standard rasterfile format (see **/usr/include/rasterfile.h**). This command is useful for viewing 8-bit rasterfiles on devices that can only display monochrome images.

OPTIONS

**–d** Use ordered dither to convert the input file instead of thresholding.

**–rgba** *threshold*

Set the threshold for the red, green, blue, and average pixel color values. Pixels whose color values are greater than or equal to all of the thresholds are given a value of 0 (white) in the output rasterfile; other pixels are set to 1 (black). The average threshold defaults to 128, the individual thresholds to zero.

EXAMPLE

The command

**example% screendump -f /dev/cgtwo | rasfilter8to1**

prints a monochromatic representation of the **/dev/cgtwo** frame buffer on the printer named **versatec** using the **v** output filter (see **/etc/printcap**).

FILES

**/usr/lib/rasfilters/*** filters for non-standard rasterfile formats
**/usr/include/rasterfile.h**

SEE ALSO

**lpr**(1), **rastrepl**(1), **screendump**(1), **screenload**(1)

*File I/O Facilities for Pixrects* in the *Pixrect Reference Manual.*

**NAME**

rastrepl – magnify a raster image by a factor of two

**SYNOPSIS**

**rastrepl** [ *infile* [ *outfile* ]]

**DESCRIPTION**

**rastrepl** reads the rasterfile *infile* (the standard input default) and converts it to the rasterfile *outfile* (the standard output default) which is twice as large in width and height. Pixel replication is used to magnify the image. The output file has the same type as the input file.

**EXAMPLES**

The following command:

**example% screendump | rastrepl | lpr**

sends a rasterfile containing the current frame buffer contents to the Versatec plotter, doubling the size of the image so that it fills a single page.

**FILES**

**/usr/lib/rasfilters/*** filters for non-standard rasterfile formats

**SEE ALSO**

**lpr**(1), **screendump**(1), **screenload**(1)

*File I/O Facilities for Pixrects* in the *Pixrect Reference Manual.*

## NAME

rcp – remote file copy

## SYNOPSIS

**rcp** [ **–p** ] *filename1 filename2*
**rcp** [ **–p –r** ] *filename...directory*

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rcp** copies files between machines. Each *filename* or *directory* argument is either a remote file name of the form:

*hostname*:*path*

or a local file name (containing no '**:**' characters, or a '/' before any '**:**'s).

If a *filename* is not a full path name, it is interpreted relative to your home directory on *hostname*. A *path* on a remote host may be quoted (using \, ", or ´) so that the metacharacters are interpreted remotely.

**rcp** does not prompt for passwords; your current local user name must exist on *hostname* and allow remote command execution by **rsh**(1C).

**rcp** handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form

*username*@*hostname*:*filename*

to use *username* rather than your current local user name as the user name on the remote host. **rcp** also supports Internet domain addressing of the remote host, so that:

*username*@*host.domain*:*filename*

specifies the username to be used, the hostname, and the domain in which that host resides. Filenames that are not full path names will be interpreted relative to the home directory of the user named *username*, on the remote host.

The destination hostname may also take the form *hostname.username*:*filename* to support destination machines that are running older versions of **rcp**.

## OPTIONS

**–p** Attempt to give each copy the same modification times, access times, and modes as the original file.

**–r** Copy each subtree rooted at *filename*; in this case the destination must be a directory.

## FILES

**.cshrc**
**.login**
**.profile**

## SEE ALSO

**ftp**(1C), **rlogin**(1C), **rsh**(1C)

## BUGS

**rcp** is meant to copy between different hosts; attempting to **rcp** a file onto itself, as with:

**myhost% rcp tmp/file myhost:/tmp/file**

results in a severely corrupted file.

**rcp** does not detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

**rcp** can become confused by output generated by commands in a **.profile**, **.cshrc**, or **.login** file on the remote host.

**rcp** requires that the source host have permission to execute commands on the remote host when doing third-party copies.

If you forget to quote metacharacters intended for the remote host you get an incomprehensible error message.

NAME

rdist – remote file distribution program

SYNOPSIS

**rdist** [ **–bhinqRvwy** ] [ **–d** *macro = value* ] [ **–f** *distfile* ] [ **–m** *host* ] ... [ *package* ... ]

**rdist** [ **–bhinqRvwy** ] **–c** *pathname* ... [*login* **@**]*hostname*[**:***destpath* ]

AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**rdist** maintains copies of files on multiple hosts. It preserves the owner, group, mode, and modification time of the master copies, and can update programs that are executing. Normally, a copy on a remote host is updated if its size or modification time differs from the original on the local host. **rdist** reads the indicated *distfile* for instructions on updating files and/or directories. If *distfile* is '–', the standard input is used. If no **–f** option is present, **rdist** first looks in its working directory for **distfile**, and then for **Distfile**, for instructions.

**rdist** updates each *package* specified on the command line; if none are given, all packages are updated according to their entries in the *distfile*.

OPTIONS

**–b** Binary comparison. Perform a binary comparison and update files if they differ, rather than merely comparing dates and sizes.

**–h** Follow symbolic links. Copy the file that the link points to rather than the link itself.

**–i** Ignore unresolved links. **rdist** will normally try to maintain the link structure of files being transfered and warn the user if all the links cannot be found.

**–n** Print the commands without executing them. This option is useful for debugging a distfile.

**–q** Quiet mode. Do not display the files being updated on the standard output.

**–R** Remove extraneous files. If a directory is being updated, remove files on the remote host that do not correspond to those in the master (local) directory. This is useful for maintaining truly identical copies of directories.

**–v** Verify that the files are up to date on all the hosts. Any files that are out of date are displayed, but no files are updated, nor is any mail sent.

**–w** Whole mode. The whole file name is appended to the destination directory name. Normally, only the last component of a name is used when renaming files. This preserves the directory structure of the files being copied, instead of flattening the directory structure. For instance, renaming a list of files such as ( **dir1/f1 dir2/f2** ) to **dir3** would create files **dir3/dir1/f1** and **dir3/dir2/f2** instead of **dir3/f1** and **dir3/f2**. When the **–w** option is used with a filename that begins with , everything except the home directory is appended to the destination name.

**–y** Younger mode. Do not update remote copies that are younger than the master copy, but issue a warning message instead.

**–d** *macro=value*

Define *macro* to have *value*. This option is used to define or override macro definitions in the distfile. *value* can be the empty string, one name, or a list of names surrounded by parentheses and separated by white space.

**–c** *pathname* ... [*login* **@**]*hostname*[**:***destpath* ]

Update each *pathname* on the named host. (Relative filenames are taken as relative to your home directory.) If the '*login* **@**' prefix is given, the update is performed with the user ID of *login*. If the '**:***destpath*' is given, the remote file is installed as that pathname.

**-f** *distfile*

Use the description file *distfile*. A '-' as the *distfile* argument denotes the standard input.

**-m** *host*

Limit which machines are to be updated. Multiple **-m** arguments can be given to limit updates to a subset of the hosts listed in the distfile.

## USAGE

### Packages

A typical package begins with a label composed of the package name followed by a colon:

*package*:

This label allows you to group any number of file-to-host and file-to-timestamp mappings into a single distribution package. If no package label appears in the distfile, the default package includes all mappings in the file.

A file-to-host mapping specifies a list of files or directories to distribute, their destination host(s), and any **rdist** primitives to use in performing the update. A mapping of this sort takes the form:

( *pathname* ... ) -> ( *hostname* ... ) *primitive* ; [*primitive* ;] ...

In this case, each *pathname* is the full pathname of a local file or directory to distribute; each *hostname* is the name of a remote host on which those files are to be copied, and *primitive* is one of the **rdist** primitive listed under *Primitives*, below. If there is only one *pathname* or *hostname*, the surrounding parentheses can be omitted. A *hostname* can also take the form:

*login@hostname*

in which case the update is performed as the user named *login*.

A file-to-timestamp mapping is used to monitor which local files are updated with respect to a local "timestamp" file. This mapping takes the form:

( *filename* ... ) :: *timestamp-file primitive* ; [*primitive* ;] ...

In this case, *timestamp-file* is the name of a file, the modification time of which is compared with each named file on the local host. If a file is newer than *time-stamp-file*, **rdist** displays a message to that effect. If there is only one *filename*, the parentheses can be omitted.

### White Space Characters

NEWLINE, TAB, and SPACE characters are all treated as white space; a mapping continues across input lines until the start of the next mapping: either a single *filename* followed by a '->' or the opening parenthesis of a filename list.

### Comments

Comments begin with # and end with a NEWLINE.

### Macros

**rdist** has a limited macro facility. Macros are only expanded in filename or hostname lists, and in the argument lists of certain primitives. Macros cannot be used to stand for primitives or their options, or the '->' or '::' symbols.

A macro definition is a line of the form:

*macro* = *value*

A macro reference is a string of the form:

**${***macro***}**

although (as with **make**(1)) the braces can be omitted if the macro name consists of just one character.

**Metacharacters**

The shell meta-characters: [, ], {, }, * and ? are recognized and expanded (on the local host only) just as they are with **csh**(1). Metacharacters can be escaped by prepending a backslash.

The character is also expanded in the same way as with **csh**, however, it is expanded separately on the local and destination hosts.

**Filenames**

File names that do not begin with / or are taken to be relative to user's home directory on each destination host. Note that they are *not* relative to the current working directory.

**Primitives**

The following primitives can be used to specify actions **rdist** is to take when updating remote copies of each file.

**install** **–bhiRvwy** [*newname*]

Copy out-of-date files and directories (recursively). If no **install** primitive appears in the package entry, or if no *newname* option is given, the name of the local file is given to the remote host's copy. If absent from the remote host, parent directories in a filename's path are created. To help prevent disasters, a non-empty directory on a target host is not replaced with a regular file or a symbolic link by **rdist**. However, when using the **–R** option, a non-empty directory is removed if the corresponding filename is completely absent on the master host. The options for **install** have the same semantics as their command line counterparts, but are limited in scope to a particular map. The login name used on the destination host is the same as the local host unless the destination name is of the format *login@host*. In that case, the update is performed under the username *login*.

**notify** *address* ...

Send mail to the indicated DARPA *address* of the form:

*user@host*

that lists the files updated and any errors that may have occurred. If an address does not contain a '*@host*' suffix, **rdist** uses the name of the destination host to complete the address.

**except** *filename* ...

Omit from updates the files named as arguments.

**except_pat***pattern* ...

Omit from updates the filenames that match each regular-expression *pattern* (see **ed**(1) for more information on regular expressions. Note that \ and $ characters must be escaped in the distfile. Shell variables can also be used within a pattern, however shell filename expansion is not supported.

**special** [*filename*] ... "*command-line*"

Specify a Bourne shell, **sh**(1) command line to execute on the remote host after each named file is updated. If no *filename* argument is present, the *command-line* is performed for every updated file, with the shell variable **FILE** set to the file's name on the local host. The quotation marks allow *command-line* to span input lines in the distfile; multiple shell commands must be separated by semicolons (;).

The default working directory for the shell executing each *command-line* is the user's home directory on the remote host.

EXAMPLE

The following sample distfile instructs **rdist** to maintain identical copies of a shared library, a shared-library initialized data file, several include files, and a directory, on hosts named **hermes** and **magus**. On **magus**, commands are executed as root. **rdist** notifies **merlin@druid** whenever it discovers that a local file has changed relative to a timestamp file.

```
HOSTS = ( hermes root@magus )

FILES = ( /usr/local/lib/libcant.so.1.1
        /usrlocal/lib/libcant.sa.1.1 /usr/local/include/{*.h}
        /usr/local/bin )

${FILES} -> ${HOSTS}
        install -R ;
${FILES} :: /usr/local/lib/timestamp
        notify merlin@druid ;
```

FILES

| | |
|---|---|
| **/tmp/rdist*** | temporary file for update lists |

SEE ALSO

**csh**(1), **ed**(1), **sh**(1), **stat**(2)

DIAGNOSTICS

A complaint about mismatch of **rdist** version numbers may really stem from some problem with starting your shell, for example, you are in too many groups.

BUGS

Source files must reside or be mounted on the local host.

There is no easy way to have a special command executed only once after all files in a directory have been updated.

Variable expansion only works for name lists; there should be a general macro facility.

**rdist** aborts on files that have a negative modification time (before Jan 1, 1970).

There should be a "force" option to allow replacement of non-empty directories by regular files or symlinks. A means of updating file modes and owners of otherwise identical files is also needed.

CAVEATS

**root** does not have its accustomed access privileges on NFS mounted file systems. Using **rdist** to copy to such a file system may fail, or the copies may be owned by user "nobody."

## NAME

refer – expand and insert references from a bibliographic database

## SYNOPSIS

**refer** [ **–ben** ] [ **–ar** ] [ **–c***string* ] [ **–k***x* ] [ **–l***m,n* ] [ **–p** *filename* ] [ **–s***keys* ] *filename*...

## AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**refer** is a preprocessor for **nroff**(1), or **troff**(1), that finds and formats references. The input files (standard input by default) are copied to the standard output, except for lines between '.[' and '.]' command lines, Such lines are assumed to contain keywords as for **lookbib**(1), and are replaced by information from a bibliographic data base. The user can avoid the search, override fields from it, or add new fields. The reference data, from whatever source, is assigned to a set of **troff** strings. Macro packages such as **ms**(7) print the finished reference text from these strings. A flag is placed in the text at the point of reference. By default, the references are indicated by numbers.

When **refer** is used with **eqn**(1), **neqn**, or **tbl**(1), **refer** should be used first in the sequence, to minimize the volume of data passed through pipes.

## OPTIONS

**–b** Bare mode — do not put any flags in text (neither numbers or labels).

**–e** Accumulate references instead of leaving the references where encountered, until a sequence of the form:

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

**–n** Do not search the default file.

**–a***r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted, all author names are reversed.

**–c***string*
Capitalize (with SMALL CAPS) the fields whose key-letters are in *string*.

**–k***x* Instead of numbering references, use labels as specified in a reference data line beginning with the characters *%x*; By default, *x* is **L**.

**–l***m,n* Instead of numbering references, use labels from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either of *m* or *n* is omitted, the entire name or date, respectively, is used.

**–p** *filename*
Take the next argument as a file of references to be searched. The default file is searched last.

**–s***keys* Sort references by fields whose key-letters are in the *keys* string, and permute reference numbers in the text accordingly. Using this option implies the **–e** option. The key-letters in *keys* may be followed by a number indicating how many such fields are used, with a + sign taken as a very large number. The default is **AD**, which sorts on the senior author and date. To sort on all authors and then the date, for instance, use the options '**–sA+T**'.

## FILES

| | |
|---|---|
| **/usr/dict/papers** | directory of default publication lists and indexes |
| **/usr/lib/refer** | directory of programs |

**SEE ALSO**

**addbib**(1), **eqn**(1), **indxbib**(1), **lookbib**(1), **nroff**(1), **roffbib**(1), **sortbib**(1), **tbl**(1), **troff**(1)

**NAME**

rev – reverse the order of characters in each line

**SYNOPSIS**

**rev** [ *filename* ] ...

**DESCRIPTION**

**rev** copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

## NAME

rlogin – remote login

## SYNOPSIS

**rlogin** [ **–L** ] [ **–8** ] [ **–e***c* ] [ **–l** *username* ] *hostname*

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rlogin** establishes a remote login session from your terminal to the remote machine named *hostname*.

Hostnames are listed in the *hosts* database, which may be contained in the **/etc/hosts** file, the Yellow Pages *hosts* database, the Internet domain name server, or a combination of these. Each host has one official name (the first name in the database entry), and optionally one or more nicknames. Either official hostnames or nicknames may be specified in *hostname*.

Each remote machine may have a file named **/etc/hosts.equiv** containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may **rlogin** from the machines listed in the remote machine's **/etc/hosts.equiv** file without supplying a password. Individual users may set up a similar private equivalence list with the file **.rhosts** in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a SPACE. An entry in a remote user's **.rhosts** file permits the user named *username* who is logged into *hostname* to rlogin to the remote machine as the remote user without supplying a password. If the name of the local host is not found in the **/etc/hosts.equiv** file on the remote machine, and the local username and hostname are not found in the remote user's .rhosts file, then the remote machine will prompt for a password. Hostnames listed in **/etc/hosts.equiv** and **.rhosts** files must be the official hostnames listed in the hosts database; nicknames may not be used in either of these files.

To counter security problems, the **.rhosts** file must be owned by either the remote user or by root.

The remote terminal type is the same as your local terminal type (as given in your environment **TERM** variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the remote login is transparent. Flow control using ^S (CTRL-S) and ^Q (CTRL-Q) and flushing of input and output on interrupts are handled properly.

## ESCAPES

Lines that you type which start with the tilde character are "escape sequences" (the escape character can be changed using the **–e** options):

**.** Disconnect from the remote host — this is not the same as a logout, because the local host breaks the connection with no warning to the remote end.

**susp** Suspend the login session (only if you are using the C shell). **susp** is your "suspend" character, usually **^Z**, (CTRL-Z), see **tty**(1).

**dsusp** Suspend the input half of the login, but output will still be seen (only if you are using the C shell). **dsusp** is your "deferred suspend" character, usually **^Y**, (CTRL-Y), see **tty**(1).

## OPTIONS

**–L** Allow the **rlogin** session to be run in "litout" mode.

**–8** Pass eight-bit data across the net instead of seven-bit data.

**–e***c* Specify a different escape character, *c*, for the line used to disconnect from the remote host.

**–l***username*

Specify a different *username* for the remote login. If you do not use this option, the remote username used is the same as your local username.

FILES

| | |
|---|---|
| **/usr/hosts/*** | for *hostname* version of the command |
| **/etc/hosts.equiv** | list of trusted hostnames with shared usernames |
| **/.rhosts** | private list of trusted hostname/username combinations |

SEE ALSO

**rsh**(1C), **stty**(1V), **tty**(1), **ypcat**(1), **hosts**(5), **named**(8c)

BUGS

This implementation can only use the TCP network service.
More of the environment should be propagated.

NAME

rm, rmdir – remove (unlink) files or directories

SYNOPSIS

**rm** [ – ] [ **–fir** ] *filename*...

**rmdir** *directory*...

DESCRIPTION

**rm**

**rm** removes (directory entries for) one or more files. If an entry was the last link to the file, the contents of that file are lost. (See **ln**(1) for more information about multiple links to files.)

To remove a file, you must have write permission in its directory; but you do not need read or write permission on the file itself. If you do not have write permission on the file and the standard input is a terminal, **rm** displays the file's permissions and waits for you to type in a response. If your response begins with y the file is deleted; otherwise the file is left alone.

**rmdir**

**rmdir** removes each named *directory*. **rmdir** only removes empty directories.

OPTIONS

– Treat the following arguments as filenames '–' so that you can specify filenames starting with a minus.

**–f** Force files to be removed without displaying permissions, asking questions or reporting errors.

**–i** Ask whether to delete each file, and, under **–r**, whether to examine each directory. Sometimes called the *interactive* option.

**–r** Recursively delete the contents of a directory, its subdirectories, and the directory itself.

SEE ALSO

**ln**(1), **su**(1)

BUGS

'**rm –r**' removes a directory and its files only if your real user ID has write permission on that directory.

DIAGNOSTICS

**rm:** *filename*: **No such file or directory**

*filename* does not exist. **rm** will also return false (1) if **rm** was invoked with '**echo $status**' and *filename* was not found.

WARNING

It is forbidden to remove the file '..' to avoid the antisocial consequences of inadvertently doing something like '**rm –r .***'.

## NAME

rmdel – remove a delta from an SCCS file

## SYNOPSIS

**/usr/sccs/rmdel** –r*SID filename*...

## DESCRIPTION

**rmdel** removes the delta specified by the *SID* from each named SCCS *filename*. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (that is, if a **p-file** (see **get**(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the **p-file**).

If a directory is named, **rmdel** behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with 's'.) and unreadable files are silently ignored. If a name of '–' is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either 1) if you make a delta you can remove it; or 2) if you own the file and directory you can remove a delta.

## FILES

| | |
|---|---|
| **x-file** | (see **delta**(1)) |
| **z-file** | (see **delta**(1)) |
| **p-file** | |

## SEE ALSO

**delta**(1), **get**(1), **help**(1), **prs**(1), **sccs**(1), **sccsfile**(5).

*Programming Utilities and Libraries*
*Source Code Control System User's Guide*

## DIAGNOSTICS

Use **help**(1) for explanations.

NAME

roffbib – format and print a bibliographic database

SYNOPSIS

**roffbib** [ **–e** ] [ **–h** ] [ **–m** *filename* ] [ **–n***p* ] [ **–o***list* ] [ **–Q** ] [ **–r***aN* ] [ **–s***N* ] [ **–T***term* ] [ **–V** ] [ **–x** ] [ *filename* ] ...

AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**roffbib** prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with *sortbib:*

**example% sortbib database | roffbib**

OPTIONS

**roffbib** accepts all options understood by **nroff**(1) except **–i** and **–q**.

**–e** Produce equally-spaced words in adjusted lines using full terminal resolution.

**–h** Use output tabs during horizontal spacing to speed output and reduce output character count. TAB settings are assumed to be every 8 nominal character widths.

**–m** *filename*
Prepend the macro file **/usr/share/lib/tmac/tmac.name** to the input files. There should be a space between the **–m** and the macro filename. This set of macros will replace the ones defined in **/usr/share/lib/tmac/tmac.bib**.

**–n***p* Number first generated page *p*.

**–o***list* Print only page numbers that appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; a final *N–* means from page *N* to end.

**-Q** Queue output for the phototypesetter. Page offset is set to 1 inch.

**–r***aN* Set register *a* (one-character) to *N*. The command-line argument **–rN1** will number the references starting at 1.

Four command-line registers control formatting style of the bibliography, much like the number registers of **ms**(7). The flag **–rV2** will double space the bibliography, while **–rV1** will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the **–rL6i** argument, and the page offset can be set from the default of 0 to one inch by specifying **–rO1i** (capital O, not zero).

**–s***N* Halt prior to every *N* pages for paper loading or changing (default *N*=1). To resume, enter NEWLINE or RETURN.

**–T** Specify *term* as the terminal type.

**–V** Send output to the Versatec. Page offset is set to 1 inch. This option not available on Sun386i systems.

**–x** If abstracts or comments are entered following the **%X** field key, **roffbib** will format them into paragraphs for an annotated bibliography. Several **%X** fields may be given if several annotation paragraphs are desired.

FILES

**/usr/share/lib/tmac/tmac.bib**
file of macros used by **nroff/troff**

SEE ALSO

**addbib**(1), **indxbib**(1), **lookbib**(1), **nroff**(1) **refer**(1), **sortbib**(1),

**refer** — *a Bibliography System* in *Formatting Documents*

BUGS

Users have to rewrite macros to create customized formats.

## NAME

rpcgen – an RPC protocol compiler

## SYNOPSIS

**rpcgen** *infile*
**rpcgen –c | –h | –l | –m** [ **–o** *outfile* ] [ *infile* ]
**rpcgen –s** *transport* [ **–o** *outfile* ] [ *infile* ]

## DESCRIPTION

**rpcgen** is a tool that generates C code to implement an RPC protocol. The input to **rpcgen** is a language similar to C known as RPC Language (Remote Procedure Call Language). Information about the syntax of RPC Language is available in the *'rpcgen' Programming Guide* in the *Network Programming* manual.

**rpcgen** is normally used as in the first synopsis where it takes an input file and generates four output files. If the *infile* is named **proto.x**, then **rpcgen** will generate a header file in **proto.h**, XDR routines in **proto_xdr.c**, server-side stubs in **proto_svc.c**, and client-side stubs in **proto_clnt.c**.

The other synopses shown above are used when one does not want to generate all the output files, but only a particular one. Their usage is described in the USAGE section below.

The C-preprocessor, **cpp**(1), is run on all input files before they are actually interpreted by **rpcgen**, so all the **cpp** directives are legal within an **rpcgen** input file. For each type of output file, **rpcgen** defines a special **cpp** symbol for use by the **rpcgen** programmer:

| | |
|---|---|
| **RPC_HDR** | defined when compiling into header files |
| **RPC_XDR** | defined when compiling into XDR routines |
| **RPC_SVC** | defined when compiling into server-side stubs |
| **RPC_CLNT** | defined when compiling into client-side stubs |

In addition, **rpcgen** does a little preprocessing of its own. Any line beginning with '**%**' is passed directly into the output file, uninterpreted by **rpcgen**.

You can customize some of your XDR routines by leaving those data types undefined. For every data type that is undefined, **rpcgen** will assume that there exists a routine with the name **xdr_** prepended to the name of the undefined type.

## OPTIONS

**–c** Compile into XDR routines.

**–h** Compile into C data-definitions (a header file)

**–l** Compile into client-side stubs.

**–m** Compile into server-side stubs, but do not generate a "main" routine. This option is useful for doing callback-routines and for people who need to write their own "main" routine to do initialization.

**–o** *outfile*
Specify the name of the output file. If none is specified, standard output is used (**–c**, **–h**, **–l** and **–s** modes only).

**–s** *transport*
Compile into server-side stubs, using the the given transport. The supported transports are **udp** and **tcp**. This option may be invoked more than once so as to compile a server that serves multiple transports.

## SEE ALSO

**cpp**(1)

*'rpcgen' Programming Guide* in *Network Programming*

**BUGS**

Nesting is not supported. As a work-around, structures can be declared at top-level, and their name used inside other structures in order to achieve the same effect.

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

## NAME

rsh – remote shell

## SYNOPSIS

**rsh** [ **–l** *username* ] [ **–n** ] *hostname command*

**rsh** *hostname* [ **–l** *username* ] [ **–n** ] *command*

*hostname* [ **–l** *username* ] [ **–n** ] *command*

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rsh** connects to the specified *hostname* and executes the specified *command*. **rsh** copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; **rsh** normally terminates when the remote command does.

If you omit *command*, instead of executing a single command, **rsh** logs you in on the remote host using **rlogin**(1C). Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. See EXAMPLES.

Hostnames are given in the *hosts* database, which may be contained in the **/etc/hosts** file, the Yellow Pages *hosts* database, the Internet domain name database, or some combination of the three. Each host has one official name (the first name in the database entry) and optionally one or more nicknames. Official hostnames or nicknames may be given as *hostname*.

If the name of the file from which **rsh** is executed is anything other than "rsh," **rsh** takes this name as its *hostname* argument. This allows you to create a symbolic link to **rsh in the name** of a host which, when executed, will invoke a remote shell on that host. The **/usr/hosts** directory is provided to be populated with symbolic links in the names of commonly used hosts. By including **/usr/hosts** in your shell's search path, you can run **rsh** by typing *hostname* to your shell.

Each remote machine may have a file named **/etc/hosts.equiv** containing a list of trusted hostnames with which it shares usernames. Users with the same username on both the local and remote machine may **rsh** from the machines listed in the remote machine's **/etc/hosts** file. Individual users may set up a similar private equivalence list with the file **.rhosts** in their home directories. Each line in this file contains two names: a *hostname* and a *username* separated by a space. The entry permits the user named *username* who is logged into *hostname* to use **rsh** to access the remote machine as the remote user. If the name of the local host is not found in the **/etc/hosts.equiv file on the remote machine,** and the local username and hostname are not found in the remote user's **.rhosts** file, then the access is denied. The hostnames listed in the **/etc/hosts.equiv** and **.rhosts** files must be the official hostnames listed in the hosts database; nicknames may not be used in either of these files.

**rsh** will not prompt for a password if access is denied on the remote machine unless the *command* argument is omited.

## OPTIONS

**–l** *username*

Use *username* as the remote username instead of your local username. In the absence of this option, the remote username is the same as your local username.

**–n** Redirect the input of **rsh** to **/dev/null**. You sometimes need this option to avoid unfortunate interactions between **rsh** and the shell which invokes it. For example, if you are running **rsh** and invoke a **rsh** in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. The **–n** option will prevent this.

The type of remote shell (**sh**, **rsh**, or other) is determined by the user's entry in the file **/etc/passwd** on the remote system.

EXAMPLES

The command:

**example% rsh lizard cat lizard.file >> example.file**

appends the remote file **lizard.file** from the machine called **lizard** to the file called **example.file** on the machine called **example**, while the command:

**example% rsh lizard cat lizard.file ">>" another.lizard.file**

appends the file **lizard.file** on the machine called **lizard** to the file **another.lizard.file** which also resides on the machine called **lizard**.

FILES

**/etc/hosts**
**/usr/hosts/***
**/etc/passwd**

SEE ALSO

**rlogin**(1C), **vi**(1), **ypcat**(1), **hosts**(5), **named**(8c)

BUGS

You cannot run an interactive command (such as **vi**(1)); use **rlogin** if you wish to do so.

Stop signals stop the local **rsh** process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

The current local environment is not passed to the remote shell.

Sometimes the **–n** option is needed for reasons that are less than obvious. For example, the command:

**example% rsh somehost dd if=/dev/nrmt0 bs=20b | tar xvpBf -**

will put your shell into a strange state. Evidently, what happens is that the **tar** terminates before the **rsh**. The **rsh** then tries to write into the "broken pipe" and, instead of terminating neatly, proceeds to compete with your shell for its standard input. Invoking **rsh** with the **–n** option avoids such incidents.

Note: this bug occurs only when **rsh** is at the beginning of a pipeline and is not reading standard input. Do not use the **–n** if **rsh** actually needs to read standard input. For example,

**example% tar cf - . | rsh sundial dd of=/dev/rmt0 obs=20b**

does not produce the bug. If you were to use the **–n** in a case like this, **rsh** would incorrectly read from **/dev/null** instead of from the pipe.

## NAME

rup – show host status of local machines (RPC version)

## SYNOPSIS

**rup** [ **–h** ] [ **–l** ] [ **–t** ] [ *host*... ]

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rup** gives a status similar to **uptime** for remote machines; It broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

When *host* arguments are given, rather than broadcasting **rup** will only query the list of specified hosts.

A remote host will only respond if it is running the **rstatd** daemon, which is normally started up from **inetd**(8C).

## OPTIONS

**–h** Sort the display alphabetically by host name.

**–l** Sort the display by load average.

**–t** Sort the display by up time.

## FILES

**/etc/servers**

## SEE ALSO

**ruptime**(1C), **inetd**(8C), **rstatd**(8C)

## BUGS

Broadcasting does not work through gateways.

## NAME

ruptime – show host status of local machines

## SYNOPSIS

**ruptime** [ **–alrtu** ]

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**ruptime** gives a status line like **uptime** for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Normally, the listing is sorted by host name, but this order can be changed by specifying one of the options listed below.

## OPTIONS

**–a** Count even those users who have been idle for an hour or more.

**–l** Sort the display by load average.

**–r** Reverse the sorting order.

**–t** Sort the display by up time.

**–u** Sort the display by number of users.

## FILES

**/var/spool/rwho/whod.***
data files

## SEE ALSO

**rup**(1C), **rwho**(1C)

## NAME

rusers – who's logged in on local machines (RPC version)

## SYNOPSIS

**rusers** [ **–ahilu** ] [ *host*... ]

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

The **rusers** command produces output similar to **users**(1) and **who**(1), but for remote machines. It broadcasts on the local network, and prints the responses it receives. Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below. When *host* arguments are given, rather than broadcasting **rusers** will only query the list of specified hosts.

The default is to print out a listing in the style of **users**(1) with one line per machine. When the **-l** flag is given, a **rwho**(1C) style listing is used. In addition, if a user has not typed to the system for a minute or more, the idle time is reported.

A remote host will only respond if it is running the **rusersd** daemon, which is normally started up from **inetd**(8C).

## OPTIONS

**–a** Give a report for a machine even if no users are logged on.

**–h** Sort alphabetically by host name.

**–i** Sort by idle time.

**–l** Give a longer listing in the style of **who**(1).

**–u** Sort by number of users.

## FILES

**/etc/servers**

## SEE ALSO

**rwho**(1C), **users**(1), **who**(1), **inetd**(8C), **rusersd**(8C)

## BUGS

Broadcasting does not work through gateways.

## NAME

rwall – write to all users over a network

## SYNOPSIS

**/usr/etc/rwall** *hostname...*
**/usr/etc/rwall –n** *netgroup...*
**/usr/etc/rwall –h hostname –n netgroup**

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rwall** reads a message from standard input until EOF. It then sends this message, preceded by the line '**Broadcast Message ...**', to all users logged in on the specified host machines. With the **-n** option, it sends to the specified network groups, which are defined in **netgroup**(5).

A machine can only receive such a message if it is running **rwalld**(8C), which is normally started up from **/etc/servers** by the daemon **inetd**(8C).

## FILES

**/etc/servers**

## SEE ALSO

**wall**(1), **netgroup**(5), **inetd**(8C), **rwalld**(8C), **shutdown**(8)

## BUGS

The timeout is fairly short in order to be able to send to a large group of machines (some of which may be down) in a reasonable amount of time. Thus the message may not get through to a heavily loaded machine.

NAME

rwho – who's logged in on local machines

SYNOPSIS

**rwho** [ **–a** ]

AVAILABILITY

The **rwho** service daemon, **rwhod**(8C) must be enabled for this command to return useful results. Refer to **finger**(1), **rup**(1C) and **rusers**(1C) for alternatives.

AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

The **rwho** command produces output similar to **who**(1), but for all machines on your network. If no report has been received from a machine for 5 minutes, **rwho** assumes the machine is down, and does not report users last known to be logged into that machine.

If a user has not typed to the system for a minute or more, **rwho** reports this idle time. If a user has not typed to the system for an hour or more, the user is omitted from the output of **rwho** unless the **–a** flag is given.

OPTIONS

**–a** Report all users whether or not they have typed to the system in the past hour.

FILES

**/var/spool/rwho/whod.***

information about other machines

SEE ALSO

**finger**(1), **rup**(1C), **ruptime**(1C), **rusers**(1C), **who**(1), **rwhod**(8C)

BUGS

Does not work through gateways.

This is unwieldy when the number of machines on the local net is large.

NAME

sact – print current SCCS file editing activity

SYNOPSIS

**/usr/sccs/sact** *filename...*

DESCRIPTION

**sact** informs the user of any SCCS files which have had one or more '**get –e**' commands applied to them, that is, there are files out for editing, and deltas are pending. If a directory is named on the command line, **sact** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of '–' is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by SPACE characters.

| | |
|---|---|
| Field 1 | Specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta. |
| Field 2 | Specifies the SID for the new delta to be created. |
| Field 3 | Contains the logname of the user who will make the delta (that is, executed a **get**(1) for editing). |
| Field 4 | Contains the date that '**get –e**' was executed. |
| Field 5 | Contains the time that '**get –e**' was executed. |

SEE ALSO

**delta**(1), **get**(1), **help**(1), **sccs**(1), **unget**(1)

*Programming Utilities and Libraries*

DIAGNOSTICS

Use **help**(1) for explanations.

## NAME

sccs – front end for the Source Code Control System (SCCS)

## SYNOPSIS

**sccs** [ **–r** ] [ **–d***prefixpath* ] [ **–p***finalpath* ] *command* [ *SCCS-flags* ... ] [ *filename* ... ]

## DESCRIPTION

The **sccs** command is a front end to the utility programs of the Source Code Control System (SCCS).

**sccs** normally prefixes each *filename*, or the last component of each *filename*, with the string 'SCCS/s.', because you normally keep your SCCS database files in a directory called **SCCS**, and each database file starts with an **'s.'** prefix. If the environment variable PROJECTDIR is set, and is an absolute pathname (that is, begins with a slash) **sccs** will search for SCCS files in the directory given by that variable. If it is a relative pathname (that is, does not begin with a slash), it is treated as the name of a user, and **sccs** will search in that user's home directory for a directory named **src** or **source**. If that directory is found, **sccs** will search for SCCS files in the directory given by that variable.

**sccs** program options must appear before the *command* argument. Flags to be passed to the actual SCCS command (utility program) must appear after the *command* argument. These flags are specific to the *command* being used, and are discussed in *Programming Utilities and Libraries*

**sccs** also includes the capability to run **'set user id'** to another user to provide additional protection. Certain commands (such as **admin**(1)) cannot be run **'set user id'** by all users, since this would allow anyone to change the authorizations. Such commands are always run as the real user.

## OPTIONS

**–r** Run **sccs** as the real user rather than as whatever effective user **sccs** is **'set user id'** to.

**–d***prefixpath*

Define the prefix portion of the pathname for the SCCS database files. The default prefix portion of the pathname is the current directory. *prefixpath* is prefixed to the entire pathname. See EXAMPLES.

This flag overrides any directory specified by the **PROJECTDIR** environment variable.

**–p***finalpath*

Define the name of a lower directory in which the SCCS files will be found; SCCS is the default. *finalpath* is appended before the final component of the pathname. See EXAMPLES.

## USAGE

### Additional sccs Commands

Several "pseudo-commands" are available in addition to the usual SCCS commands. These are:

**create** **create** is used when creating new **s.** files. For example, given a C source language file called **'obscure.c'**, **create** would perform the following actions: (1) create the **'s.'** file called **'s.obscure.c'** in the SCCS directory; (2) rename the original source file to **',obscure.c'**; (3) do an **'sccs get'** on **'obscure.c'**. Compared to the SCCS **admin** command, **create** does more of the startup work for you and should be used in preference to **admin**.

**enter** **enter** is just like **create**, except that it does not do the final **'sccs get'**. It is usually used if an **'sccs edit'** is to be performed immediately after the **enter**.

**edit** Get a file for editing.

**delget** Perform a **delta** on the named files and then **get** new versions. The new versions have ID keywords expanded, and so cannot be edited.

**deledit** Same as **delget**, but produces new versions suitable for editing. **deledit** is useful for making a "checkpoint" of your current editing phase.

**fix** Remove the named delta, but leaves you with a copy of the delta with the changes that were in it. **fix** must be followed by a **–r** flag. **fix** is useful for fixing small compiler bugs, etc. Since **fix** does not leave audit trails, use it carefully.

**clean** Remove everything from the current directory that can be recreated from SCCS files. **clean** checks for and does not remove any files being edited. If '**clean –b**' is used, branches are **not** checked to see if they are currently being edited. Note: **–b** is dangerous if you are keeping the branches in the same directory.

**unedit** "Undo" the last **edit** or '**get –e**' and return a file to its previous condition. If you **unedit** a file being edited, all changes made since the beginning of the editing session are lost.

**info** Display a list of all files being edited. If the **–b** flag is given, branches (that is, SID's with two or fewer components) are ignored. If the **–u** flag is given (with an optional argument), only files being edited by you (or the named user) are listed.

**check** Check for files currently being edited, like **info**, but returns an exit code rather than a listing: nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. **check** may thus be included in an "install" entry in a makefile, to ensure that everything is included in an SCCS file before a version is installed.

**tell** Display a list of files being edited on the standard output. Filenames are separated by NEWLINE characters. Take the **–b** and **–u** flags like **info** and **check**.

**diffs** Compare (in **diff**-like format) the current version of the program you have out for editing and the versions in SCCS format. **diffs** accepts the same arguments as **diff**, except that the **–c** flag must be specified as **–C** instead, because the **–c** flag is taken as a flag to **get** indicating which version is to be compared with the current version.

**print** Print verbose information about the named files. **print** does an '**sccs prs –e**' followed by an '**sccs get –p –m**' on each file.

EXAMPLES

The command:

**example% sccs –d/usr/include get sys/inode.h**

converts to:

**get /usr/include/sys/SCCS/s.inode.h**

The intent here is to create aliases such as:

**alias syssccs sccs –d/usr/src**

which will be used as:

**example% syssccs get cmd/who.c**

The command:

**example% sccs –pprivate get usr/include/stdio.h**

converts to:

**get usr/include/private/s.stdio.h**

To put a file called **myprogram.c** into SCCS format for the first time, assuming also that there is no SCCS directory already existing:

**example% mkdir SCCS**
**example% sccs create myprogram.c**
**myprogram.c:**
**1.1**
**14 lines**
*after you have verified that everything is all right*
*you remove the version of the file that starts with a comma:*
**example% rm ,myprogram.c**
**example%**

To get a copy of **myprogram.c** for editing, edit that file, then place it back in the SCCS database:

```
example% sccs edit myprogram.c
1.1
new delta 1.2
14 lines
example% vi myprogram.c
your editing session
example% sccs delget myprogram.c
comments? Added abusive responses for compatibility with rest of system
1.2
7 inserted
7 deleted
7 unchanged
1.2
14 lines
example%
```

To *get* a file from another directory:

```
example% sccs -p/usr/src/sccs/ get cc.c
```

or:

```
example% sccs get /usr/src/sccs/cc.c
```

To make a delta of a large number of files in the current directory:

```
example% sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
example% sccs info -b
```

To *delta* everything that you are editing:

```
example% sccs delta `sccs tell -u`
```

In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
        sccs get $(REL) $@
```

**Regular sccs Commands**

The "regular" SCCS commands are described very briefly below. It is unlikely that you ever need to use these commands because the user interface is so complicated, and the **sccs** front end command does 99.9% of the interesting tasks for you.

**admin** Create new SCCS files and changes parameters of existing SCCS files. You can use 'sccs **create**' to create new SCCS files, or use '**sccs admin**' to do other things.

**cdc** Change the commentary material in an SCCS delta.

**comb** Combine SCCS deltas and reconstructs the SCCS files.

**delta** Permanently introduces changes that were made to a file previously retrieved using 'sccs **get**'. You can use '**sccs delget**' as the more useful version of this command since '**sccs delget**' does all of the useful work and more.

**get** Extract a file from the SCCS database, either for compilation, or for editing when the **-e** option is used. Use '**sccs get**' if you really need it, but '**sccs delget**' will normally have done this job for you. Use **sccs edit** instead of **get** with the **-e** option.

**help** Supposed to help you interpret SCCS error messages, but usually just parrots the message

and is generally not considered very helpful.

**prs** Display information about what is happening in an SCCS file.

**rmdel** Remove a delta from an SCCS file.

**sccsdiff** Compare two versions of an SCCS file and generates the differences between the two versions.

**val** Determine if a given SCCS file meets specified criteria. If you use the **sccs** command, you should not need to use **val**, because its user interface is unbelievable.

**what** Display SCCS identification information.

FILES

**/usr/sccs/***

SEE ALSO

**admin**(1), **cdc**(1), **comb**(1), **delta**(1), **get**(1), **help**(1), **prs**(1), **rmdel**(1), **sact**(1), **sccsdiff**(1), **unget**(1), **val**(1), **what**(1), **sccsfile**(5)

*Programming Utilities and Libraries*

## NAME

sccsdiff – compare two versions of an SCCS file

## SYNOPSIS

**/usr/sccs/sccsdiff** **–r** *SID1* **–r** *SID2* [ **–p** ] [ *–diffopts* ] *filenames*

## DESCRIPTION

**sccsdiff** compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but options apply to all files.

## OPTIONS

**–r***SID?* *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to **diff**(1) in the order given.

**–p** Pipe output for each file through **pr**(1V).

*–diffopts*

The **–c**, **–e**, **–f**, **–h**, **–b** and **–D** options of **diff** can be specified here.

## FILES

**/tmp/get?????** temporary files

## SEE ALSO

**diff**(1), **get**(1), **help**(1), **pr**(1V), **sccs**(1)

*Programming Utilities and Libraries*

## DIAGNOSTICS

*filename*: **No differences**

If the two versions are the same.

Use **help**(1) for explanations.

NAME

screenblank – turn off the screen when the mouse and keyboard are idle

SYNOPSIS

**screenblank** [ **–m** ] [ **–k** ] [ **–d** *interval* ] [ **–e** *interval* ] [ **–f** *frame-buffer* ]

DESCRIPTION

**screenblank** turns off the display when the mouse and keyboard are idle for an extended period (the default is 10 minutes). **screenblank** will continue to run until killed by hand, using '**kill –9** *processid*' or by logging out.

OPTIONS

**–m** Do not check whether the mouse has been idle.

**–k** Do not check whether the keyboard has been idle.

**–d** *interval*

Disable after *interval* seconds. *interval* is a number of the form *xxx.xxx* where each *x* is a decimal digit. The default is 600 seconds (10 minutes).

**–e** *interval*

Enable within *interval* seconds. *interval* is the time between successive polls for keyboard or mouse activity. If a poll detects keyboard or mouse activity, the display is resumed. *interval* is a number of seconds, of the form *xxx.xxx* where each *x* is a decimal digit. The default is 0.25 seconds.

**–f** *frame-buffer*

*frame-buffer* is the path name of the frame-buffer on which video disabling/enabling applies. The defaults is **/dev/fb**.

FILES

**/dev/fb**

SEE ALSO

**lockscreen**(1), **sunview**(1)

BUGS

When not running **sunview**(1), only the RETURN key resumes video display.

NAME

screendump – dump a frame-buffer image to a file

SYNOPSIS

**screendump** [ **–ceo8** ] [ **–f** *frame-buffer* ] [ **–t** *type* ] [ **–xyXY** *value* ] [ *filename* ]

DESCRIPTION

**screendump** reads the contents of a frame buffer and writes the display image to *filename* (the default is the standard output) in Sun standard rasterfile format.

If the frame buffer has both an overlay plane and color planes, **screendump** examines the overlay enable plane and tries to make the output file represent what is visible on the screen. It maps the overlay plane foreground and background colors into the closest values present in the color map for the color planes.

OPTIONS

**–c** Dump the frame-buffer contents directly without making a temporary copy in a memory pixrect. Saves time and memory but lengthens the time the frame-buffer must be inactive to guarantee a consistent screen dump.

**–e** Set the output rasterfile type to 2, RT_BYTE_ENCODED. For most images this saves a significant amount of space compared to the standard format.

**–o** Dump the frame-buffer overlay plane only (ignored if the display does not have an overlay plane).

**–8** Dump the frame-buffer color planes only (ignored if the display does not have color planes).

**–f** *frame-buffer*
Dump the specified frame-buffer device (default is **/dev/fb**).

**–t** *type* Set the output rasterfile type (default 1, RT_STANDARD).

**–x** *value*
**–y** *value*
Set the x or y coordinate of the upper left corner of the area to be dumped to the given value.

**–X** *value*
**–Y** *value*
Set the width or height of the area to be dumped to the given value.

EXAMPLES

The command:

**example% screendump save.this.image**

writes the current contents of the console frame buffer into the file **save.this.image,**

while the command:

**example% screendump –f /dev/cgtwo save.color.image**

writes the current contents of the color frame buffer **/dev/cgtwo** into the file **save.color.image.**

The command:

**example% screendump | lpr –Pversatec –v**

sends a rasterfile containing the current frame-buffer to the line printer, selecting the printer **versatec** and the **v** output filter (see **/etc/printcap**).

FILES

| | |
|---|---|
| **/usr/include/rasterfile.h** | definition of rasterfile format |
| **/usr/lib/rasfilters/*** | filters for non-standard rasterfile formats |
| **/dev/fb** | default frame buffer device |
| **/etc/printcap** | |

SEE ALSO

**lpr**(1), **rastrepl**(1), **screenload**(1), **rasfilter8to1**(1)

*Pixrect Reference Manual*

BUGS

The output file or the screen may be corrupted if the frame-buffer contents are modified while the dump is in progress.

NAME

screenload – load a frame-buffer image from a file

SYNOPSIS

**screenload** [ **–dopr** ] [ **–f***frame-buffer* ] [ **–xyXY** *value* ] [ **–bgnw** ] [ **–h** *count data* ... ] [ **–i** *color* ] [ *filename* ]

DESCRIPTION

**screenload** reads a Sun standard rasterfile (see **rasterfile**(5)) and displays it on a frame-buffer. **screenload** is able to display monochrome images on a color display, but cannot display color images on a monochrome display. If the input file contains a color image, a frame-buffer has not been explicitly specified, and **/dev/fb** is a monochrome frame-buffer, **screenload** looks for a color frame-buffer with one of the standard device names.

If the image contained in the input file is larger than the actual resolution of the display, **screenload** clips the right and bottom edges of the image. If the input image is smaller than the display (for example, loading an 1152-by-900 image on a 1600-by-1280 high resolution display), **screenload** centers the image on the display surface and fills the border area with solid black (by default). Various options may be used to change the image location, or to change or disable the fill pattern.

OPTIONS

**–d** Print a warning message if the display size does not match the rasterfile image size.

**–o** Load the image on the overlay plane of the display (ignored if the display does not have an overlay plane).

**–p** Wait for a NEWLINE to be typed on the standard input before exiting.

**–r** Reverse the foreground and background of the output image. Useful when loading a screendump made from a reverse video screen.

**–f***frame-buffer*
Display the image on the specified frame-buffer device (default **/dev/fb**).

**–x** *value*
**–y** *value*
Set the x or y coordinate of the upper left corner of the image on the display to the given value.

**–X** *value*
**–Y** *value*
Set the maximum width or height of the displayed image to the given value.

**–b** Fill the border area with a pattern of solid ones (default). On a monochrome display this results in a black border; on a color display the color map value selected by the **–i** option determines the border color.

**–g** Fill the border area with a pattern of "desktop grey". On a monochrome display this results in a border matching the default background pattern used by SunView; on a color display the color map value selected by the **–i** option determines the foreground border color, though the pattern is the same as on a monochrome display.

**–n** Do not fill the border area.

**–w** Fill the border area with a pattern of solid zeros. On a monochrome display this results in a white border; on a color display the color map value at index 0 determines the border color.

**–h** *count data* ...
Fill the border area with the bit pattern described by the following *count* 16-bit hexadecimal constants. Note: a "1" bit is black and a "0" bit is white on the monochrome display; on a color display the color map value selected by the **–i** option determines the border foreground color. The number of hex constants in the pattern is limited to 16.

**–i** *color* Fill the border area with the given color value (default 255).

EXAMPLES

The command:

**example% screenload saved.display.image**

loads the raster image contained in the file **saved.display.image** on the display type indicated by the rasterfile header in that file.

**example% screenload –f /dev/cgtwo monochrome.image**

reloads the raster image in the file **monochrome.image** on the color frame-buffer device **/dev/cgtwo.**

The command:

**example% screenload –h1 ffff small.saved.image**

is equivalent to the **–b** option (fill border with black), while

**example% screenload –h4 8888 8888 2222 2222 small.saved.image**

is equivalent to the **–g** option (fill border with desktop grey).

FILES

| | |
|---|---|
| **/usr/include/rasterfile.h** | definition of rasterfile format |
| **/usr/lib/rasfilters/*** | filters for non-standard rasterfile formats |
| **/dev/fb** | default frame buffer device |

SEE ALSO

**rasfilter8to1**(1), **rastrepl**(1), **screendump**(1), **screenload**(1)

*Pixrect Reference Manual*

NAME

script – make typescript of a terminal session

SYNOPSIS

**script** [ **–a** ] [ *filename* ]

DESCRIPTION

**script** makes a typescript of everything printed on your terminal. The typescript is written to *filename*, or appended to *filename* if the **–a** option is given. It can be sent to the line printer later with **lpr**(1). If no file name is given, the typescript is saved in the file **typescript.**

The script ends when the forked shell exits.

OPTIONS

**–a** Append the script to the specified file instead of writing over it.

SEE ALSO

**lpr**(1)

BUGS

**script** places *everything* in the log file. This is not what the naive user expects.

NAME

sdiff – contrast two text files by displaying them side-by-side

SYNOPSIS

**sdiff** [ **–l** ] [ **–o** *outfile* ] [ **–s** ] [ **–w** *n* ] *filename1 filename2*

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**sdiff** uses the output of **diff** to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *filename1*, a > in the gutter if the line only exists in *filename2*, and a | for lines that are different. See EXAMPLES.

OPTIONS

**–w** *n* Use *n* as the width of the output line. The default line length is 130 characters.

**–l** Only print the left side of any identical lines.

**–s** Silent. Do not print identical lines.

**–o** *outfile*

Use the next argument, *output*, as the name of an output file created as an interactively controlled merging of *filename1* and *filename2*. Identical lines of *filename1* and *filename2* are copied to *output*. Sets of differences, as produced by **diff**, are printed; where a set of differences share a common gutter character. After printing each set of differences, **sdiff** prompts with a **%** and waits for you to type one of the following commands:

**l** Append the left column to the output file.

**r** Append the right column to the output file.

**s** Turn on silent mode; do not print identical lines.

**v** Turn off silent mode.

**e l** Call the **ed**(1) with the left column.

**e r** Call **ed**(1) with the right column.

**e b** Call **ed**(1) with the concatenation of left and right columns.

**e** Call **ed**(1) with a zero length file.

On exit from **ed**(1), the resulting file is concatenated to the named output file.

**q** Exit from the program.

EXAMPLES

A sample output of **sdiff** would look like this:

```
x    |    y
a         a
b    <
c    <
d         d
     >    c
```

SEE ALSO

**diff**(1), **ed**(1)

## NAME

sed – stream editor

## SYNOPSIS

**sed** [ **–n** ] [ **–e** *script* ] [ **–f** *sfilename* ] [ *filename* ]...

## DESCRIPTION

**sed** copies the *filename*s (standard input default) to the standard output, edited according to a script of commands.

## OPTIONS

**–n** Suppress the default output.

**–e** *script*

*script* is an edit command for **sed**. If there is just one **–e** option and no **–f** options, the **–e** flag may be omitted.

**–f** *sfilename*

Take the script from *sfilename*.

## USAGE

### sed Scripts

**sed scripts** consist of editing commands, one per line, of the following form:

[ *address* [, *address* ] ] *function* [ *arguments* ]

In normal operation **sed** cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), sequentially applies all commands with *addresses* matching that pattern space until reaching the end of the script, copies the pattern space to the standard output (except under **–n**), and finally, deletes the pattern space.

Some commands use a *hold space* to save all or part of the pattern space for subsequent retrieval.

An *address* is either:

a decimal number linecount, which is cumulative across input files;

a **$**, which addresses the last input line;

or a context address, which is a */regular expression/* in the style of **ed**(1);

with the following exceptions:

\?RE? In a context address, the construction \ *?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note: in the context address **\xabc\xdefx**, the second x stands for itself, so that the regular expression is **abcxdef**.

**\n** Matches a NEWLINE embedded in the pattern space.

**.** Matches any character except the NEWLINE ending the pattern space.

*null* A command line with no address selects every pattern space.

*address* Selects each pattern space that matches.

*address1* , *address2*

Selects the inclusive range from the first pattern space matching *addrress1* to the first pattern space matching *address2*. Selects only one line if *address1* is greater than or equal to *address2*.

**Comments**

If the first nonwhite character in a line is a '#' (pound sign), **sed** treats that line as a comment, and ignores it. If, however, the first such line is of the form:

**#n**

**sed** runs as if the **–n** flag were specified.

**Functions**

The maximum number of permissible addresses for each function is indicated in parentheses in the list below.

An argument denoted *text* consists of one or more lines, all but the last of which end with \ to hide the NEWLINE. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial SPACE and TAB characters against the stripping that is done on every script line.

An argument denoted *rfilename* or *wfilename* must terminate the command line and must be preceded by exactly one SPACE. Each *wfilename* is created before processing begins. There can be at most 10 distinct *wfilename* arguments.

(1)**a**
*text* Append: place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the ':' command bearing the *label*. Branch to the end of the script if *label* is empty.

(2) **c**
*text* Change: delete the pattern space. With 0 or 1 address or at the end of a 2 address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first NEWLINE. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**
*text* Insert: place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) **p** Print: copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first NEWLINE to the standard output.

(1) **q** Quit: branch to the end of the script. Do not start a new cycle.

(2) **r** *rfilename*
Read the contents of *rfilename*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of '/'. For a fuller description see **ed**(1). *flags* is zero or more of:

*n* *n*= 1 – 512. Substitute for just the *n*th occurrence of the *regular* expression.

**g** Global: substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

**p** Print the pattern space if a replacement was made.

**w** *wfilename*
Write: append the pattern space to *wfilename* if a replacement was made.

(2) **t** *label* Test: branch to the ':' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfilename*
Write: append the pattern space to *wfilename*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/
Transform: replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function* Do not: apply the *function* (or group, if *function* is '{') only to lines *not* selected by the address(es).

(0) **:** *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to. Note: the maximum length of *label* is seven characters.

(1) = Place the current line number on the standard output as a line.

(2) **{** Execute the following commands through a matching '}' only when the pattern space is selected.

(0) An empty command is ignored.

**System V sed Scripts**
Initial SPACE and TAB characters are *not* stripped from text lines.

DIAGNOSTICS

**Too many commands**
The command list contained more than 200 commands.

**Too much command text**
The command list was too big for **sed** to handle. Text in the **a**, **c**, and **i** commands, text read in by **r** commands, addresses, regular expressions and replacement strings in **s** commands, and translation tables in **y** commands all require **sed** to store data internally.

**Command line too long**
A command line was longer than 4000 characters.

**Too many line numbers**
More than 256 decimal number linecounts were specified as addresses in the command list.

**Too many files in w commands**
More than 10 different files were specified in **w** commands or **w** options for **s** commands in the command list.

**Too many labels**
More than 50 labels were specified in the command list.

**Unrecognized command**
A command was not one of the ones recognized by **sed**.

**Extra text at end of command**
A command had extra text after the end.

**Illegal line number**
An address was neither a decimal number linecount, a **$**, nor a context address.

**Space missing before filename**
There was no space between a **r** or **w** command, or the **w** option for a **s** command, and the filename specified for that command.

**Too many {'s**
There were more { than } in the list of commands to be executed.

**Too many }'s**
There were more } than { in the list of commands to be executed.

**No addresses allowed**
A command that takes no addresses had an address specified.

**Only one address allowed**
A command that takes one address had two addresses specified.

**"\digit" out of range**
The number in a *n* item in a regular expression or a replacement string in a **s** command was greater than 9.

**Bad number**
One of the endpoints in a range item in a regular expression (that is, an item of the form {*n*} or {*n*,*m*}) was not a number

**Range endpoint too large**
One of the endpoints in a range item in a regular expression was greater than 255.

**More than 2 numbers given in \{ \}**
More than two endpoints were given in a range expression.

**} expected after **
A \ appeared in a range expression and was not followed by a }.

**First number exceeds second in \{ \}**
The first endpoint in a range expression was greater than the second.

**Illegal or missing delimiter**
The delimiter at the end of a regular expression was absent.

**\( \) imbalance**
There were more \( than \), or more \) than \(, in a regular expression.

**[ ] imbalance**
There were more [ than ], or more ] than [, in a regular expression.

**First RE may not be null**
The first regular expression in an address or in a **s** command was null (empty).

**Ending delimiter missing on substitution**
The ending delimiter in a **s** command was absent.

**Ending delimiter missing on string**
The ending delimiter in a **y** command was absent.

**Transform strings not the same size**
The two strings in a **y** command were not the same size.

**Suffix too large - 512 max**
The suffix in a **s** command, specifying which occurrence of the regular expression should be replaced, was greater than 512.

**Label too long**
A label in a command was longer than 8 characters.

**Duplicate labels**
The same label was specified by more than one **:** command.

**File name too long**
The filename specified in a **r** or **w** command, or in the **w** option for a **s** command, was longer than 1024 characters.

**Output line too long.**
An output line was longer than 4000 characters long.

**Too many appends or reads after line *n***
More than 20 **a** or **r** commands were to be executed for line *n*.

**Hold space overflowed.**
More than 4000 characters were to be stored in the *hold space*.

SEE ALSO

**awk**(1), **ed**(1), **lex**(1), **grep**(1V)

*Editing Text Files*

BUGS

There is a combined limit of 200 **–e** and **–f** arguments. In addition, there are various internal size limits which, in rare cases, may overflow. To overcome these limitations, either combine or break out scripts, or use a pipeline of **sed** commands.

NAME

selection_svc – SunView selection service

SYNOPSIS

**selection_svc** [ **–d** ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

The SunView selection service handles the state and rank of various selections for its client programs. A selection service is started automatically by **sunview**(1). However, you can also start one explicitly using the ®command.

OPTIONS

**–d** Debug a client program. Use the alternate socket provided for testing and debugging a client program. The second selection service invoked with the **–d** handles a different set of selections, and can co-exist with one started in the normal fashion. Each service responds only to requests directed to its own socket. The client to be debugged can be directed to use the "debugging" service by using the **seln_use_test_service()** procedure, as described in the *SunView 1 System Programmer's Guide.*

SEE ALSO

**sunview**(1)

*SunView 1 Beginner's Guide*

*SunView 1 System Programmer's Guide*

## NAME

sh – shell, the standard UNIX system command interpreter and command-level language

## SYNOPSIS

**sh** [ **–acefhiknstuvx** ] [ *arguments* ]

## DESCRIPTION

**sh**, the Bourne shell, is the standard UNIX-system command interpreter. It executes commands read from a terminal or a file.

### Definitions

A *blank* is a TAB or a SPACE character. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, –, $, and ! .

### Invocation

If the shell is invoked through **execve**(2), **execv**(3), or **execl**(3), and the first character of argument zero is '–', commands are initially read from **/etc/profile** and from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **sh**.

## OPTIONS

The flags below are interpreted by the shell on invocation only; unless the **–c** or **–s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters for use with the commands that file contains.

**–i** If the –i flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **'kill 0'** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

**–s** If the –s flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for **Special Commands**) is written to file descriptor 2.

**–c** *string*
If the **–c** flag is present commands are read from *string*.

The remaining flags and arguments are described under the **set** command, under **Special Commands**, below.

## USAGE

Refer to *Doing More with SunOS: Beginner's Guide* for more information about using the shell as a programming language.

### Commands

A *simple command* is a sequence of nonblank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see **execve**(2)). The *value* of a *simple command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see **sigvec**(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by '| ' (or, for historical compatibility, by '^'). The standard output of each command but the last is connected by a **pipe** (see **pipe**(2)) to the standard input of the next command. Each command is run as a separate process; the shell normally waits for the last command to terminate before prompting for or accepting the next input line. The exit status of a pipeline is the exit status of its last command.

A *list* is a sequence of one or more *simple command*s or pipelines, separated by ';', '&', '&&', or '| | ', and optionally terminated by ';' or '&'. Of these four symbols, ';' and '&' have equal precedence, which is lower than that of ' && ' and '| | '. The symbols ' && ' and '| | ' also have equal precedence. A semicolon (;) sequentially executes the preceding pipeline; an ampersand (&) asynchronously executes the preceding pipeline (the shell does *not* wait for that pipeline to finish). The symbols && and| | are used to indicate conditional execution of the list that follows. With **&& , *list*** is executed only if the preceding

pipeline (or command) returns a zero exit status. With | |, *list* is executed only if the preceding pipeline (or command) returns a nonzero exit status. An arbitrary number of NEWLINE characters may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple command* or one of the following constructions. Unless otherwise stated, the value returned by a command is that of the last *simple command* executed in the construction.

**for** *name* [ **in** *word* ... ]
**do** *list* **done**
Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see **Parameter Substitution** below). Execution ends when there are no more words in the list.

**case** *word* **in** [*pattern*[ | *pattern* ] ... ) *list* ;; ] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see **Filename Generation**) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

**if** *list* **then** *list* [ **elif** *list* **then** *list* ] ... [ **else** *list* ] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

**while** *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

( *list* ) Execute *list* in a subshell.

{ *list*;} *list* is simply executed.

*name* () {*list*;}
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see **Execution**).

The following words are only recognized as the first word of a command and when not quoted:

**if then else elif fi case esac for while until do done { }**

**Comments**

A word beginning with # and all the following characters up to a NEWLINE are ignored.

**Command Substitution**

The shell reads commands from the string between two grave accents (``) and the standard output from these commands may be used as all or part of a word. Trailing NEWLINE characters from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ... ` ... ` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a NEWLINE character (\NEWLINE), both the backslash and the NEWLINE are removed (see **Quoting**, later). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, `, ", NEWLINE, and $ are left intact when the command string is read.

**Parameter Substitution**

The character **$** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

*name*=*value* [ *name*=*value* ] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

**${***parameter***}**

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is '*' or '@', all the positional parameters, starting with **$1**, are substituted (separated by SPACE characters). Parameter **$0** is set from argument zero when the shell is invoked.

If the colon (:) is omitted from the following expressions, the shell only checks whether *parameter* is set or not.

**${***parameter***:-***word***}**

If *parameter* is set and is nonnull, substitute its value; otherwise substitute *word*.

**${***parameter***:=***word***}**

If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

**${***parameter***:?***word***}**

If *parameter* is set and is nonnull, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message **'parameter null or not set'** is printed.

**${***parameter***:+***word***}**

If *parameter* is set and is nonnull, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

**echo ${***d***:-'pwd'}**

The following parameters are automatically set by the shell:

| | |
|---|---|
| **#** | The number of positional parameters in decimal. |
| **-** | Flags supplied to the shell on invocation or by the **set** command. |
| **?** | The decimal value returned by the last synchronously executed command. |
| **$** | The process number of this shell. |
| **!** | The process number of the last background command invoked. |

The following parameters are used by the shell:

**HOME** The default argument (home directory) for the **cd** command.

**PATH** The search path for commands (see **Execution** below).

**CDPATH**

The search path for the **cd** command.

**MAIL** If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

**MAILCHECK**

This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

**MAILPATH**

A colon (:) separated list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename can be followed by **%** and a message that will be printed when the modification time changes. The default message is **'you have mail'**.

**PS1** Primary prompt string, by default **'$ '**.

**PS2** Secondary prompt string, by default **'> '**.

**IFS** Internal field separators, normally SPACE, TAB, and NEWLINE.

**SHELL** When the shell is invoked, it scans the environment (see **Environment** below) for this name.

The shell gives default values to **PATH, PS1, PS2, MAILCHECK** and **IFS. HOME** and **MAIL** are set by **login**(1).

**Blank Interpretation**

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

**Input/Output**

A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple command* or may precede or follow a *command* and are *not* passed on to the invoked command. Note: parameter and command substitution occurs before *word* or *digit* is used.

*<word* Use file *word* as standard input (file descriptor 0).

*>word* Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

*>>word* Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the EOF); otherwise, the file is created.

*<<[-]word* After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an EOF. If, however, '-' is appended to:

- leading TAB characters are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),
- leading TAB characters are stripped from the shell input as it is read and before each line is compared with *word*, and
- shell input is read up to the first line that literally matches the resulting *word*, or to an EOF.

If any character of *word* (see **Quoting**, later), no additional processing is done to the shell input. If no characters of *word* are quoted:

- parameter and command substitution occurs,
- (escaped) \NEWLINE is ignored, and
- '\' must be used to quote the characters '\', '$', and '`'.

The resulting document becomes the standard input.

*<&digit* Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using *>&digit*.

*<&-* The standard input is closed. Similarly for the standard output using '>&-'.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (namely, file *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under **Commands**, if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by & the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

**Filename Generation**

Before a command is executed, each command *word* is scanned for the characters '*', '?', and '['. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename is found that matches the pattern, the word is left unchanged. The character '.' at the start of a filename or immediately following a '/', as well as the character '/' itself, must be matched explicitly.

| | |
|---|---|
| * | Matches any string, including the null string. |
| ? | Matches any single character. |
| [...] | Matches any one of the enclosed characters. A pair of characters separated by '–' matches any character lexically between the pair, inclusive. If the first character following the opening [ is a ! any character not enclosed is matched. |

**Quoting**

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > NEWLINE SPACE TAB

A character may be *quoted* (made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks (' ' or " "). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair \NEWLINE is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "'").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$*** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("**$1 $2** ..."); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("**$1**" "**$2**" ... ). \ quotes the characters \, ', ", and **$**. The pair \NEWLINE is removed before parameter and command substitution. If a backslash precedes characters other than \, ', ", $, and NEWLINE, then the backslash itself is quoted by the shell.

**Prompting**

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a NEWLINE is typed and further input is needed to complete a command, the secondary prompt (the value of **PS2**) is issued.

**Environment**

The *environment* (see **environ**(5V)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also '**set** –**a**'). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or

additions, all of which must be noted in **export** commands.

The environment for any *simple command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

**TERM=450** *cmd*

and

**(export TERM; TERM=450;** *cmd*)

are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set –k
echo a=b c
```

**Signals**

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

**Execution**

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the **Special Commands** listed below, it is executed in the shell process. If the command name does not match a **Special Command**, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, .... are set to the arguments of the function. If the command name matches neither a **Special Command** nor the name of a defined function, a new process is created and an attempt is made to execute the command using **execve**(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/usr/ucb:/bin:/usr/bin** (specifying **/usr/ucb**, **/bin**, and **/usr/bin**, in addition to the current directory). Directories are searched in order. The current directory is specified by a null path name, which can appear immediately after the equal sign (**PATH=:...**), between the colon delimiters (**...::...**) anywhere else in the path list, or at the end of the path list (**...:**). If the command name contains a / the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an binary executable (see **a.out**(5) for details) or an executable script (with a first line beginning with **#!**) it is assumed to be a file containing shell commands, and a subshell is spawned to read it. A parenthesized command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary **execs** later). If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the '**hash –r**' command is executed (see below).

**Special Commands**

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

**:** No effect; the command does nothing. A zero exit code is returned.

**.** *filename*
Read and execute commands from *filename* and return. The search path specified by **PATH** is used to find the directory containing *filename*.

**break** [ *n* ]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

**continue** [ *n* ]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*'th enclosing loop.

**cd***[ arg ]*
Change the current directory to *argument*. The shell parameter **HOME** is the default *argument*. The shell parameter **CDPATH** defines the search path for the directory containing *argument*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note: the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *argument* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *argument*.

**echo** [ *argument* ... ]
Echo arguments. See **echo**(1V) for usage and description.

**eval**[ *argument* ... ]
The arguments are read as input to the shell and the resulting command(s) executed.

**exec** [ *argument* ... ]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, modify the shell's input/output.

**exit** [ *n* ]
Exit a shell with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an EOF will also cause the shell to exit.)

**export** [ *name* ... ]
The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

**getopts** Use in shell scripts to parse positional parameters and check for legal options. See **getopts**(1) for usage and description.

**hash** [ **−r** ] [ *name* ... ]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **−r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *hits* is the number of times a command has been invoked by the shell process. *cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *cost* will be incremented when the recalculation is done.

**login** [ *argument* ... ]
Equivalent to '**exec login** *argument*...' See **login**(1) for usage and description.

**newgrp** [ *argument* ... ]
Equivalent to '**exec newgrp** *argument*...' See **newgrp**(1) for usage and description.

**pwd** Print the current working directory. See **pwd**(1) for usage and description.

**read** [ *name* ... ]
One line is read from the standard input and, using the internal field separator, IFS (normally SPACE or TAB), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using \NEWLINE. Characters other than NEWLINE can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is **0** unless an EOF is encountered.

**readonly** [ *name* ... ]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

**return** [ *n* ]
Exit a function with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

**set** [ **–aefhkntuvx–** [ *argument* ... ] ]
**–a** Mark variables which are modified or created for export.
**–e** Exit immediately if a command exits with a nonzero exit status.
**–f** Disable filename generation.
**–h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
**–k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
**–n** Read commands but do not execute them.
**–t** Exit after reading and executing one command.
**–u** Treat unset variables as an error when substituting.
**–v** Print shell input lines as they are read.
**–x** Print commands and their arguments as they are executed.
**—** Do not change any of the flags; useful in setting **$1** to '–'.
Using '+' rather than '–' turns off these flags. These flags can also be used upon invocation of the shell. The current set of flags may be found in '$–'. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, and so on. If no arguments are given, the values of all names are printed.

**shift** [ *n* ]
The positional parameters are shifted to the left, from position *n*+1 to position 1, and so on. Previous values between **$1** and **$***n* are discarded. If *n* is not given, it is assumed to be 1.

**test** Evaluate conditional expressions. See **test**(1V) for usage and description.

**times** Print the accumulated user and system times for processes run from the shell.

**trap** [ *arg* ] [ *n* ] ...
The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note: *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

**type** [ *name* ... ]
For each *name*, indicate how it would be interpreted if used as a command name.

**umask** [ *ooo* ]
The user file-creation mode mask is set to *ooo* (see **csh**(1)). The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively. The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file. For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**). The current value of the mask is printed if *ooo* is omitted.

**unset** [ *name* ... ]
For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS** cannot be unset.

**wait** [ *n* ]
Wait for the background process whose process ID is *n* and report its termination status. If *n* is omitted, all the shell's currently active background processes are waited for and the return code will be zero.

EXIT STATUS

Errors detected by the shell, such as syntax errors, return a nonzero exit status. If the shell is being used noninteractively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

**/etc/profile**
**$HOME/.profile**
**/tmp/sh***
**/dev/null**

SEE ALSO

**csh**(1), **cd**(1), **echo**(1V), **env**(1), **getopts**(1), **login**(1), **newgrp**(1), **pwd**(1), **test**(1V), **wait**(1), **dup**(2), **fork**(2), **execve**(2), **pipe**(2), **sigvec**(2), **wait**(2), **execl**(3), **a.out**(5), **environ**(5V)

*Doing More with SunOS: Beginner's Guide*

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to **exec** the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For **wait** *n*, if *n* is not an active process ID, all the shell's currently active background processes are waited for and the return code will be zero.

WARNINGS

Words used for filenames in input/output redirection are not interpreted for filename generation (see **File Name Generation**, above). For example, '**cat file1 >a***' will create a file named '**a***'.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message '**cannot fork, too many processes**', try using the **wait**(1) command to clean up your background processes. If this does not help, the system process table is probably full or you have too many active foreground processes. There is a limit to the number of process IDs associated with your login, and to the number the system can keep track of.

NAME

shelltool – run a shell (or other program) in a SunView terminal window

SYNOPSIS

**shelltool** [ **–C** ] [ **–B** *boldstyle* ] [ **–I** *command* ] [ *generic-tool-arguments* ] [ *program* [ *arguments* ] ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**shelltool** is a standard SunView facility for shells or other programs that may use a standard tty-based interface.

When invoked, **shelltool** runs a program, (usually a shell) in an interactive terminal emulator based on a tty subwindow. Keyboard input is passed to that program. If the program is a shell, it accepts commands and runs programs in the usual way.

**cmdtool**(1), which provides moused-based editing, logging, and scrolling capabilities, also supports shell-level programs. See *SunView 1 Beginner's Guide* for more information.

To run graphics programs, use **gfxtool**(1).

OPTIONS

**–C** Redirect system console output to this **shelltool**.

**–B** *boldstyle*

Set the style for displaying bold text to *boldstyle*. *boldstyle* can be a string specifying one of the choices for the **/Tty/Bold_style** default, see **Defaults Options**, below, or it may be a numerical value for one of those choices, from 0 to 8, corresponding to the placement of the choice in the list.

**–I** *command*

Pass *command* to the shell. SPACE characters within the command must be escaped.

*generic-tool-arguments*

**shelltool** accepts the generic tool arguments listed in **sunview**(1).

If a *program* argument is present, **shelltool** runs it. If no *program* is given, **shelltool** runs the program indicated by the SHELL environment variable, or **/usr/bin/sh** by default.

USAGE

**Defaults Options**

These options are available through **defaultsedit**(1).

**/Tty/Bold_style**

Select a style for emphasized text:

| | |
|---|---|
| **None** | Disable emphasis. |
| **Offset_X** | Thicken characters horizontally. |
| **Offset_Y** | Thicken characters vertically. |
| **Offset_X_and_Y** | Thicken characters both horizontally and vertically. |
| **Offset_XY** | Thicken characters diagonally. |
| **Offset_X_and_XY** | Thicken character both horizontally and diagonally. |
| **Offset_Y_and_XY** | Thicken characters both vertically and diagonally. |
| **Offset_X_and_Y_and_XY** | Thicken characters horizontally, vertically and diagonally. |
| **Invert** | Display emphasis as inverse video (the standard default). |

**/Tty/Inverse_mode**

Select a style for inverse video display:

| | |
|---|---|
| **Enable** | Enable inverse mode for inverted text. |
| **Disable** | Disable inverse mode for inverted text. |

**Same_as_bold** Display inverted text as bold text.

**/Tty/Underline_mode**

Select a style for underlined text:

| | |
|---|---|
| **Enable** | Enable underline mode for underlined text. |
| **Disable** | Disable underline mode for underlined text. |
| **Same_as_bold** | Display underlined text as bold text. |

**/Tty/Retained**

When set to "Yes", hidden tty subwindow areas are retained in memory. This enhances the speed of repainting the screen at the expense of memory area. "No" is the standard default; it specifies that tty subwindows are not retained.

**The Terminal Emulator**

The tty subwindow is a terminal emulator. Whenever a tty subwindow is created, the startup file **/.ttyswrc** is read for initialization parameters that are specific to the tty subwindow.

**The .ttyswrc File**

A sample **.ttyswrc** file can be found in **/usr/lib/ttyswrc**. The command format for this file is:

| | |
|---|---|
| **#** | Comment. |
| **set** *variable* | Turn on the specified variable. |
| **mapi** *key text* | When *key* is typed pretend *text* was input. |
| **mapo** *key text* | When *key* is typed pretend *text* was output. |

The only currently defined variable is **pagemode.** *key* is one of L1-L15, F1-F15, T1-T15, R1-R15, LEFT, or RIGHT (see note below). *text* may contain escapes such as \E, \n, ^X, etc. (ESCAPE, NEWLINE, and CTRL-X, respectively). See **termcap**(5) for the format of the string escapes that are recognized. Note: **mapi** and **mapo** may be replaced by another keymapping mechanism in the future.

When using the default kernel keyboard tables, the keys L1, LEFT, RIGHT, BREAK, R8, R10, R12, and R14 cannot be mapped in this way; they send special values to the tty subwindow. Also, when using the default kernel keyboard tables, L1-L10 are now used by SunView. See **input_from_defaults**(1) and **kbd**(4S) for more information on how to change the behavior of the keyboard.

It is possible to have terminal-based programs drive the tool in which its tty subwindow resides by sending special escape sequences. These escape sequences may also be sent by typing a key appropriately mapped using the **mapo** function described above. The following functions pertain to the tool in which the tty subwindow resides, not the tty subwindow itself.

| | |
|---|---|
| **\E[1t** | – open |
| **\E[2t** | – close (become iconic) |
| **\E[3t** | – move, with interactive feedback |
| **\E[3;TOP;LEFTt** | – move, to **TOP LEFT** (pixel coordinates) |
| **\E[4t** | – stretch, with interactive feedback |
| **\E[4;WIDTH;HTt** | – stretch, to **WIDTH HT** size (in pixels) |
| **\E[5t** | – front |
| **\E[6t** | – back |
| **\E[7t** | – refresh |
| **\E[8;ROWS;COLSt** | – stretch, to **ROWS COLS** size (in characters) |
| **\E[11t** | – report if open or iconic by sending **\E[1t** or **\E[2t** |
| **\E[13t** | – report position by sending **\E[3;TOP;LEFTt** |
| **\E[14t** | – report size in pixels by sending **\E[4;WIDTH;HTt** |
| **\E[18t** | – report size in characters by sending **\E[8;ROWS;COLSt** |
| **\E[20t** | – report icon label by sending **\E]Llabel\E** |
| **\E[21t** | – report tool header by sending **\E]llabel\E** |
| **\E]l<text>\E** | – set tool header to **<text>** |
| **\E]I<file>\E** | – set icon to the icon contained in **<file>**; **<file>** must be in *iconedit* output format |

| | |
|---|---|
| **\E]L<label>\E** | – set icon label to **<label>** |
| **\E[>OPT;...h** | – turn SB OPT on (**OPT** = 1 => pagemode), for example, **\E[>1;3;4h** |
| **\E[>OPT;...k** | – report **OPT**; sends **\E[>OPTl** or **\E[>OPTh** for each **OPT** |
| **\E[>OPT;...l** | – turn **OPT** off (**OPT** = 1 => pagemode), for example, **\E[>1;3;4l** |

See EXAMPLES for an example of using this facility.

**Selections**

Terminal subwindows support a selection facility that allows you to capture a block of text, move it between windows, and replicate it. You can make a selection by clicking the left button on the mouse at the top-left character of the block to capture, and then clicking the middle button on the bottom-right character. The selected text is highlighted. Multiple clicks of the LEFT mouse button capture:

| | |
|---|---|
| 1 click | a character |
| 2 clicks | a word |
| 3 clicks | a line |
| 4 clicks | a screenful |

You can also make a selection by moving the mouse while holding the select button, and then releasing it. The selection is deselected if you type any key or new output is written to the window that holds the selection.

**Menu**

To manipulate your selection, press the menu button over the terminal subwindow. A *ttysw* menu appears with the menu items discussed below:

**Copy, then Paste** When there is a selection in any window, the entire item is active. Selecting it copies the selection both to the clipboard and to the insertion point (cursor). It copies selections in tty, text, command, and panel subwindows, and It is intended to bridge the gap between **Stuff** and the selection facility (see *SunView 1 Beginner's Guide*. When there is no selection but there is text on the clipboard, only **Paste** is active. In this case, the contents of the clipboard are copied to the insertion point (cursor). When there is no selection and nothing on the clipboard, this item is inactive.

**Enable Page Mode**
**Disable Page Mode** Toggle page mode on and off. Page mode prevents output from scrolling off the screen. It is an alternative to **more**(1). When page mode is on, the cursor changes to resemble a tiny stop-sign when ever a screenful of output is displayed. To restart output, type any key, or select the **Continue** menu item that temporarily replaces **Enable Page Mode**.

**Stuff** is provided for backward compatibility. It copies the selection to the insertion point (cursor) as though they had been typed from the keyboard. **Stuff** can only handle selections made in a tty subwindow.

**Flush Input** Occasionally the input buffer fills up and the terminal emulator appears to freeze. If this happens, the '**Flush Input**' appears in the menu; choosing it clears the buffer and allows you to continue.

EXAMPLES

The following aliases can be put into your **/.cshrc** file:

```
# dynamically set the name stripe of the tool:
alias header 'echo -n "\E]l\!*\E\"'
# dynamically set the label on the icon:
alias iheader 'echo -n "\E]L\!*\E\"'
# dynamically set the image on the icon:
alias icon 'echo -n "\E]I\!*\E]\"'
```

**FILES**

**/.ttyswrc**
**/usr/lib/ttyswrc**
**/usr/bin/shelltool**
**/usr/bin/sunview**
**/usr/demo/***

**SEE ALSO**

**cmdtool**(1), **defaultsedit**(1), **gfxtool**(1), **input_from_defaults**(1), **more**(1), **rlogin**(1C), **sunview**(1), **kbd**(4S), **termcap**(5)

*SunView 1 Beginner's Guide*

**BUGS**

If more than 256 characters are input to a terminal emulator subwindow without an intervening NEWLINE, the terminal emulator may hang. If this occurs, an alert will come up with a message saying '**Too many keystrokes in input buffer**'. Choosing the **Flush Input Buffer** menu item may correct the problem. This is a bug for a terminal emulator subwindow running on top of or **rlogin**(1C) to a machine with pre-4.0 release kernel.

NAME

size – display the size of an object file

SYNOPSIS

**size** [ *object-file* ... ]

Sun386i SYNOPSIS

**/usr/bin/size** [**–n**] [**–f**] [**–o**] [**–x**] [**–V**] *filename* ...

DESCRIPTION

**size** prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in hex and decimal, of each *object-file* argument. If no file is specified, **a.out** is used.

Sun386i DESCRIPTION

The Sun386i version of the System V compatibility package includes **/usr/bin/size**, which allows the System V options to be used, and creates the same output as the System V **size**(1) command; it produces section size information in bytes for each loaded section in COFF files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is given, it displays the information for all archive members.

Sun386i OPTIONS

**–n** Includes NOLOAD sections in the size.

**–f** Produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.

**–o** Print numbers in octal, instead of the default which is decimal.

**–x** Print numbers in hexadecimal.

**–V** Supply version information.

Sun386i CAVEATS

Since the size of bss sections is not known until link-edit time, this command does not give the true total size of pre-linked objects.

Sun386i DIAGNOSTICS

**size:** *name*: **cannot open**

*name* cannot be read.

**size:** *name*: **bad magic**

*name* is not an appropriate common object file.

SEE ALSO

**cc**(1V), **size**(1), **a.out**(5), **coff**(5), **ar**(5)

**NAME**

sleep – suspend execution for a specified interval

**SYNOPSIS**

**sleep** *time*

**DESCRIPTION**

**sleep** suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

**SEE ALSO**

**sleep**(3)

**BUGS**

*time* must be less than 2,147,483,647 seconds.

## NAME

snap – SunView application for system and network administration

## SYNOPSIS

**snap**

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

**snap** simplifies the execution of a variety of system administration tasks in the user-friendly environment of a SunView window. **snap** eases the following tasks: personal or centralized backup and restoration of files, management of users accounts and user groups, software installation, network administration, and management of devices such as printers, terminals, modems, and the peripheral box containing disk and tape drives.

Anyone can use **snap**, but the operations allowed depend on the secondary group membership of the user at the time that snap is invoked. There are four secondary user groups specifically recognized by **snap**, membership in which bestows various powers over the corresponding area of system administration. These are:

**accounts** users and user groups.

**devices** printers, terminals, modems, and peripheral box.

**operator** centralized backup and restoration of files, and installation of software.

**networks** domains and systems, including the New User Accounts feature and Automatic System Installation features.

A user's **snap** privileges depend upon which of these four groups he or she belongs to. If they get an account through New User Accounts, or if an administrator adds them using the defaults, new users become members of the primary group *users*, and are given all **snap** privileges. This can be changed by changing the secondary group membership of the primary group *users* with **snap**. Note: this does not change the group membership of existing users, but only of new users. The secondary group membership of existing users must be changed individually.

### Accounts

An administrator using **snap** can create new user accounts and remove existing ones, change a user's **snap** privileges, and control users' access to their accounts. New users can create their own accounts as they first login if the New User Accounts feature is activated as described under Networks below.

### Devices

Epson and Epson-like printers (most printers using the Centronics parallel interface), text serial printers, and HP Laserjet and compatible printers can be administered with **snap** . The supported terminal types are **vt-100** and **wyse.** The supported modem types are Hayes Smartmodem or a modem that is compatible with Hayes Smartmodem. For all other types of terminals, modems, or printers, the software must be configured manually. See *System and Network Administration* for details.

**snap** can add or remove, display and change information about, or disable or enable either a printer, a terminal, a modem, or the peripheral box containing disk and tape drives. Devices not added using **snap** can not be manipulated with **snap**.

### Operator

Regardless of the primary or secondary group membership of users, they can backup and restore their own files with **snap**.

Backup and removal of all files can be done by members of the **operator** group.

### Networks

Much of the network setup must be done when the first machine in the network, the master server, is started up, and when each client is connected and booted for the first time. Some of this information can never be changed.

Once the master and slave servers are installed, **snap** can be used to add and assign diskless clients to servers, remove them, modify their network roles, and perform all the functions listed above under Accounts, Devices, and Operator on any system in the network.

If desired, you can also enable or disable the feature that allows a user to create his own account while logging in (New User Accounts), and the automatic system installation feature, two possible security loopholes.

SEE ALSO

*Sun386i System and Network Administration*
*System and Network Administration*

NAME

soelim – resolve and eliminate .so requests from nroff or troff input

SYNOPSIS

**soelim** [ *filename* ... ]

AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**soelim** reads the specified files or the standard input and performs the textual inclusion implied by the **nroff**(1) directives of the form

**.so** *somefile*

when they appear at the beginning of input lines. This is useful since programs such as **tbl**(1) do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of '–' is taken to be a file name corresponding to the standard input.

Note: inclusion can be suppressed by using ' ´' instead of '.', that is,

**´ so /usr/share/lib/tmac.s**

EXAMPLE

A sample usage of **soelim** would be

**soelim exum?.n | tbl | nroff**

SEE ALSO

**colcrt**(1), **more**(1), **nroff**(1), **tbl**(1)

NAME

sort – sort and collate lines

SYNOPSIS

**sort** [ **–bdfiMnr** ] [ **–t***c* ] [ **sort-field** ...] [ **–cmu** ] [ **–o**[ ]*outputfile* ] [ **–T** *directory* ]
[ **–y** *kmem* ] [ **–z** *recsz* ] *filename*...

SYSTEM V SYNOPSIS

**/usr/5bin/sort** [ **–bdfiMnr** ] [ **–t***c* ] [ **sort-field** ...] [ **–cmu** ] [ **–o**[ ]*outputfile* ] [ **–T** *directory* ]
[ **–y** *kmem* ] [ **–z** *recsz* ] *filename*...

DESCRIPTION

The **sort** program sorts and collates lines contained in the named files, and writes the result onto the standard output. If no *filename* argument is given, or if '–' appears as an argument, **sort** accepts input from the standard input.

Output lines are normally sorted on a character-by-character basis, from left to right within a line. The default collating sequence is the ASCII character set. Lines can also be sorted according to the contents of one or more fields specified by a **sort-field**, specification, using the *+sw* (starting-word), *–ew* (end-at-word), and the –t*c* (set-TAB-character/word delimiter) options, as described under OPTIONS below. When no word delimiter is specified, one or more adjacent white-space characters (SPACE and TAB) signify the end of the previous word; the lines:

```
^^^ xyz
^^^   xyz
```

are collated as:

```
^^^   xyz
^^^ xyz
```

Each **sort-field** is evaluated in command-line order; later fields are applied to the sorting sequence only when all earlier fields compare equally. When all specified fields compare equally between two or more lines, that subset of lines is sorted on a character-by-character basis, from left to right.

SYSTEM V DESCRIPTION

When no fields are specified in the command line, the System V version of **sort** treats leading blanks as significant, even with the **–n** (numeric collating sequence) option; the lines:

```
123
  23
```

are collated as:

```
  23
123
```

OPTIONS

**Collating Flags**

**–b** Ignore leading SPACE characters when determining the starting and ending positions of a field.

**–d** Dictionary order. Only letters, digits and the white-space characters SPACE and TAB are significant in comparisons.

**–f** Fold in lower case. Treat upper- and lower-case letters equally in collating comparisons.

**–i** Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

**–M** Month order. The first three non-blank characters of the field are folded to upper case and collated according to the sequence: **JAN FEB ... DEC**. Field values outside this range appear earlier than **JAN**. The **–M** option implies the **–b** option.

**–n** Numeric collating sequence. An initial numeric string, consisting of optional blanks, optional minus signs, and zero or more digits with an optional decimal point, is sorted by arithmetic value.

The **–n** option implies the **–b** option, but only when at least one **sort-field** is specified on the command line.

**–r** Reverse the current collating sequence.

**Field Specification Options**

**–t***c* Use *c* as the word delimiter character; unlike white-space characters, adjacent delimiters indicate word breaks; if **:** is the delimiter character, **::** delimits an empty word.

**sort-field**

This is a combination of options that specifies a field, within each line, to sort on. A **sort-field** specification can take either of the following forms:

*+sw*[*cf*]
*+sw –ew*[*cf*]

where *sw* is the number of the starting word (beginning with '0') to include in the field, *ew* is the number of the word before which to end the field, and *cf* is a string containing collating flags (without a leading '–'.) When included in a **sort-field** specification, these flags apply only to the field being specified, and when given, override other collating flags given in separate arguments (which otherwise apply to an entire line).

If the *–ew* option is omitted, the field continues to the end of a line.

You can apply a character offset to *sw* and *ew* to indicate that a field is to start or end a given number of characters within a word, using the notation: '*w.c*'. A starting position specified in the form: '*+w.c*' indicates the character in position *c* (beginning with **0** for the first character), within word *w* (**1** and **1.0** are equivalent). An ending position specified in the form: '*–w.c*' indicates that the field ends at the character just prior to position *c* (beginning with **0** for the delimiter just prior to the first character), within word *w*. If the **–b** flag is in effect, *c* is counted from the first non-white-space or non-delimiter character in the field, otherwise, delimiter characters are counted.

**Other Options**

**–c** Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

**–m** Merge only, the input files are already sorted.

**–u** Unique. Emit only the first line in each set of lines for which all sorting fields compare equally.

**–o***outputfile*
**–o** *outputfile*

Direct output to the file specified as *outputfile*, instead of the standard output. This file may be the same as one of the input files.

**–y** *kmem*

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, **sort** begins using a system default memory size, and continues to add space as needed. If this option is given **sort** starts with *kmem*, kilobytes of memory, if allowed, or as close to that amount as possible. Supplying **–y0** guarantees that **sort** starts with a minimum of memory. By convention, **–y** (with no argument) starts with maximum memory.

**–z** *recsz*

The size of the longest line read is recorded in the sort phase so that buffers can be allocated during the merge phase. If the sort phase is omitted because either of the **–c** or **–m** options is in effect, a default size of 1024 bytes is used. Lines longer than the buffer size terminate **sort** abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) avoids this.

**–T** *directory*

The *directory* argument is the name of a directory in which to place temporary files.

EXAMPLES

Sort the contents of **infile** with word number 1 (the second word) as the sort key:

**sort +1 –2 infile**

Sort, in reverse order, the contents of **infile1** and **infile2**, placing the output in **outputfile** and using the first character of the second field as the sort key:

**sort –r –o outputfile +1.0 –1.1 infile1 infile2**

Sort, in reverse order, the contents of **infile1** and **infile2** using the first non-blank character of the second field as the sort key:

**sort –r +1.0b –1.1b infile1 infile2**

Print the password file (**passwd**(5)) sorted by the numeric user ID (the third colon-separated field):

**sort –t: +2n –3 /etc/passwd**

Print the lines of the already sorted file **infile**, suppressing all but the first occurrence of lines having the same third field (the options **–mu** with just one input file make the choice of a unique representative from a set of equal lines predictable):

**sort –mu +2 –3 infile**

FILES

**/usr/tmp/stm???**

SEE ALSO

**comm**(1), **join**(1), **rev**(1), **uniq**(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (such as when input lines are too long), and for disorders discovered under the **–c** option.

When the last line of an input file is missing a NEWLINE, **sort** appends one, prints a warning message, and continues.

NAME

sortbib – sort a bibliographic database

SYNOPSIS

**sortbib** [ –s*KEYS* ] *database*...

AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**sortbib** sorts files of records containing **refer** key-letters by user-specified keys. Records may be separated by blank lines, or by '.[' and '.]' delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so **sortbib** may not be used in a pipeline to read standard input.

By default, **sortbib** alphabetizes by the first **%A** and the **%D** fields, which contain the senior author and date. The –s option is used to specify new *KEYS*. For instance, –**sATD** will sort by author, title, and date, while –**sA+D** will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

**sortbib** sorts on the last word on the **%A** line, which is assumed to be the author's last name. A word in the final position, such as '**jr.**' or '**ed.**', will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the **nroff** convention '**\0**' in place of a blank. A **%Q** field is considered to be the same as **%A**, except sorting begins with the first, not the last, word. **sortbib** sorts on the last word of the **%D** line, usually the year. It also ignores leading articles (like '**A**' or '**The**') when sorting by titles in the **%T** or **%J** fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, **sortbib** places that record before other records containing that field.

SEE ALSO

**addbib**(1), **indxbib**(1), **lookbib**(1), **refer**(1), **roffbib**(1)

**refer** in *Formatting Documents*

BUGS

Records with missing author fields should probably be sorted by title.

## NAME

spell, spellin, spellout – report spelling errors

## SYNOPSIS

**spell** [ **–bvx** ] [ **–d** *hlist* ] [ **–s** *hstop* ] [ **–h** *spellhist* ] [ *filename* ] ...

**spellin** [ *inlist* ]

**spellout** [ **–d** ] *outlist*

## DESCRIPTION

**spell** collects words from the named files, and looks them up in a hashed spelling list derived from the system dictionary. Words that do not appear in the list, or cannot be derived from those that do appear by applying certain inflections, prefixes or suffixes, are displayed on the standard output.

If there are no *filename* arguments, words to check are collected from the standard input. **spell** ignores most **troff**(1), **tbl**(1), and **eqn**(1) constructs. Copies of all output words are accumulated in the history file, and a *stop* list filters out misspellings (for example, thier=thy–y+ier) that would otherwise pass.

The standard spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

**spellin** adds words from the standard input to a hashed spelling list, and writes the resulting hashed list to the standard output. If an *inlist* argument is supplied, the input words are hashed together with that existing spelling list. If not, **spellin** creates a new list from scratch.

**spellout** looks up each word from the standard input, compares them with *outlist*, and displays those that are missing from that list. With the **–d** option, **spellout** displays input words that appear in the list.

## OPTIONS

**–b** Check British spelling. Besides preferring "centre", "colour", "travelled", and so on, this option insists upon *-ise* in words like *standardize*, despite what Fowler and the OED say.

**–v** Print all words not literally in the spelling list, as well as plausible derivations from spelling list words.

**–x** Print every plausible stem with '=' for each word.

**–d** *hlist* Use the file *hlist* as the hashed spelling list.

**–s** *hstop*
Use *hstop* as the hashed stop list.

**–h** *spellhist*
Place misspelled words with a user/date stamp in file *spellhist*.

## FILES

| | |
|---|---|
| **/usr/dict/hlist[ab]** | hashed spelling lists, American & British |
| **/usr/dict/hstop** | hashed stop list |
| **/usr/dict/words** | system dictionary—list of properly spelled words and roots |
| **/usr/lib/spell** | program called by the **/usr/bin/spell** shell script |

## NOTE

Misspelled words can be monitored by default by setting the **H** variable in **/usr/bin/spell** to the name of a file that has permission mode 666.

## SEE ALSO

**deroff**(1), **sed**(1V), **sort**(1V), **tee**(1)

BUGS

The spelling list's coverage is uneven; new installations may wish to monitor the output for several months to gather local additions.

British spelling was done by an American.

NAME

spline – interpolate smooth curve

SYNOPSIS

**spline** [ **–aknpx** ] ...

DESCRIPTION

**spline** takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by **graph**(1G).

OPTIONS

**–a** Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

**–k** The constant $k$ used in the boundary value computation

$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$

is set by the next argument. By default $k = 0$.

**–n** Space output points so that approximately $n$ intervals occur between the lower and upper $x$ limits. (Default $n = 100$.)

**–p** Make output periodic, that is, match derivatives at ends. First and last input values should normally agree.

**–x** Next 1 (or 2) arguments are lower (and upper) $x$ limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

**graph**(1G)

R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed.

DIAGNOSTICS

When data is not strictly monotonic in $x$, **spline** reproduces the input without interpolating extra points.

BUGS

A limit of 1000 input points is enforced silently.

**NAME**

split – split a file into pieces

**SYNOPSIS**

**split** [ *–number* ] [ *infile* [ *outfile* ] ]

**DESCRIPTION**

**split** reads *infile* and writes it in *n*-line pieces (default 1000) onto a set of output files (as many files as necessary). The name of the first output file is *outfile* with **aa** appended, the second file is *outfile* **ab**, and so on lexicographically.

If no *outfile* is given, **x** is used as default (output files will be called **xaa**, **xab**, etc.).

If no *infile* is given, or if '–' is given in its stead, then the standard input file is used.

**OPTIONS**

*–number*

Number of lines in each piece.

**NAME**

strings – find printable strings in an object file or binary

**SYNOPSIS**

**strings** [ – ] [ **–o** ] [ *–number* ] *filename* . . .

**DESCRIPTION**

**strings** looks for ASCII strings in a binary file. A string is any sequence of 4 or more printing characters ending with a NEWLINE or a NULL.

**strings** is useful for identifying random object files and many other things.

**OPTIONS**

– Look everywhere in the file for strings. If this flag is omitted, **strings** only looks in the initialized data space of object files.

**–o** Precede each string by its offset in the file.

*–number*
Use *number* as the minimum string length rather than 4.

**SEE ALSO**

**od**(1V)

**BUGS**

The algorithm for identifying strings is extremely primitive.

NAME

strip – remove symbols and relocation bits from an object file

SYNOPSIS

**strip** *filename...*

DESCRIPTION

**strip** removes the symbol table and relocation bits ordinarily attached to the output of the assembler and linker. This is useful to save space after a program has been debugged.

The effect of **strip** is the same as use of the **–s** option of **ld**(1).

SEE ALSO

**ld**(1), **sun3cvt**(1), **a.out**(5)

BUGS

Unstripped 2.0 binary files will not run if stripped by the 3.0 version. A message of the form:

**pid** *xxx*: **killed due to swap problems in I/O error mapping page.**

when attempting to run a program indicates that this is the problem.

## NAME

stty – set or alter the options for a terminal

## SYNOPSIS

**stty** [ **–ag** ] [ *option* ] ...

## SYSTEM V SYNOPSIS

**/usr/5bin/stty** [ **–ag** ] [ *option* ] ...

## DESCRIPTION

sets certain terminal

**stty** I/O options for the device that is the current standard output. Without arguments, it reports the settings of certain terminal options for the device that is the standard output; the settings are reported on the standard error.

Detailed information about the modes listed in the first five groups below may be found in **termio**(4). Options in the last group are implemented using options in the previous groups. Note: many combinations of options make no sense, but no sanity checking is performed.

## SYSTEM V DESCRIPTION

**stty** sets or reports terminal options for the device that is the current standard input; the settings are reported on the standard output.

## OPTIONS

**–a** Report all of the option settings.

**–g** Report current settings in a form that can be used as an argument to another **stty** command.

### Special Requests

**speed** The terminal speed alone is printed on the standard output.

**size** The terminal (window) sizes are printed on the standard output, first rows and then columns.

### Control Modes

[–]**parenb** Enable parity generation and detection. With a '–', disable parity checking.

[–]**parodd** Select odd parity. With a '–', select even parity.

**cs5 cs6 cs7 cs8**
Select character size.

**0** Hang up phone line immediately.

**50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 exta 38400 extb**
Set terminal baud rate to the number given, if possible. (Not all speeds are supported by all hardware interfaces.)

[–]**hupcl** Hang up connection on last close. With a '–', do not hang up connection.

[–]**hup** Same as **hupcl**.

[–]**cstopb** Use two stop bits per character. With a '–', use one stop bit per character.

[–]**cread** Enable the receiver. With a '–', disable the receiver.

[–]**clocal** Assume a line without modem control. With a '–', assume a line with modem control.

### Input Modes

[–]**ignbrk** Ignore break on input. With a '–', do not ignore a break on input.

[–]**brkint** Signal SIGINT on break. With a '–', do not signal.

[–]**ignpar** Ignore parity errors. With a '–', do not ignore parity errors.

[–]**parmrk** Mark parity errors With a '–', do not mark parity errors.

[–]**inpck** Enable input parity checking. With a '–', disable input parity checking.

[–]**istrip** Strip input characters to seven bits. With a '–', do not strip input characters.

[–]**inlcr** Map NEWLINE to RETURN on input. With a '–', do not map on input.

[–]**igncr** Ignore RETURN on input. With a '–', do not ignore RETURN on input.

[–]**icrnl** Map RETURN to NEWLINE on input. With a '–', do not map.

[–]**iuclc** Map upper-case alphabetics to lower case on input. With a '–', do not map.

[–]**ixon** Enable **START/STOP** output control. With a '–', disable output control. When enabled, output is stopped by sending a **STOP** character and started by sending a **START** character.

[–]**ixany** Allow any character to restart output. With a '–', only restart with a **START** character.

[–]**decctlq** Same as **–ixany.**

[–]**ixoff** Request that the system send **START/STOP** characters when the input queue is nearly empty/full. With a '–', request that the system not send **START/STOP** characters.

[–]**tandem** Same as **ixoff.**

[–]**imaxbel** Request that the system send a BEL character to your terminal, and not to flush the input queue, if a character received when the input queue is full. With a '–', request that it flush the input queue and not send a BEL character.

**Output Modes**

[–]**opost** Post-process output. With a '–', do not post-process output; ignore all other output modes.

[–]**olcuc** Map lower-case alphabetics to upper case on output. With a '–', do not map.

[–]**onlcr** Map NEWLINE to RETURN-NEWLINE on output. With a '–', do not map.

[–]**ocrnl** Map RETURN to NEWLINE on output. With a '–', do not map.

[–]**onocr** Do not place RETURN characters at column zero. With a '–', do place RETURN characters at column zero.

[–]**onlret** On the terminal NEWLINE performs the RETURN function. With a '–', NEWLINE does not perform the RETURN function.

[–]**ofill** Use fill characters for delays. With a '–', use timing for delays.

[–]**ofdel** Fill characters are DEL characters. With a '–', fill characters are NUL characters.

**cr0 cr1 cr2 cr3**
Select style of delay for RETURN characters.

**nl0 nl1** Select style of delay for LINEFEED characters.

**tab0 tab1 tab2 tab3**
Select style of delay for horizontal TAB characters.

**bs0 bs1** Select style of delay for BACKSPACE characters.

**ff0 ff1** Select style of delay for form FORMFEED characters.

**vt0 vt1** Select style of delay for vertical TAB characters.

**Local Modes**

[–]**isig** Enable the checking of characters against the special characters **INTR** and **QUIT**. With a '–', disable this checking.

[–]**icanon** Enable canonical input (**ERASE**, **KILL**, **WERASE**, and **RPRNT** processing). With a '–', disable canonical input.

[–]**cbreak** Same as **–icanon.**

[–]**xcase** Perform canonical upper/lower-case presentation. With a '–', do not perform canonical upper/lower-case presentation.

[–]**echo** Echo back every character typed. With a '–', do not echo back.

[–]**echoe** Echo the **ERASE** character as a sequence of BACKSPACE-SPACE-BACKSPACE. With a '–', echo the **ERASE** character as itself.

[–]**crterase** Same as **echoe.**

[–]**echok** Echo NEWLINE after echoing a **KILL** character. With a '–', do not echo NEWLINE after echoing a **KILL** character.

**lfkc** Same as **echok**; obsolete.

[–]**echonl** Echo NEWLINE, even if **echo** is not set. With a '–', do not echo NEWLINE if **echo** is not set.

[–]**noflsh** Disable flush after **INTR** or **QUIT**. With a '–', enable flush.

[–]**tostop** Stop background jobs that attempt to write to the terminal. With a '–', allow background jobs to write to the terminal.

[–]**echoctl** Echo control characters as x (and delete as '?'.) Print two BACKSPACE characters following the **EOF** character (default CTRL-D). With a '–', echo control characters as themselves.

[–]**ctlecho** Same as **echoctl.**

[–]**echoprt** Echo erased characters backwards within '\' and '/'; used on printing terminals. With a '–', echo erased characters as indicated by **echoe.**

[–]**prterase** Same as **echoprt.**

[–]**echoke** Echo the **KILL** character by erasing each character on the line as indicated by **echoprt** and **echoe.** With a '–', echo the **KILL** character as indicated by **echoctl** and **echok.**

[–]**crtkill** Same as **echoke.**

*control-character c*
Set *control-character* to *c*, where *control-character* is one of **erase, kill, intr, quit, eof, eol, eol2, start, stop, susp, rprnt, flush, werase,** or **lnext.** If *c* is preceded by a caret (^), (escaped from the shell) then the value used is the corresponding CTRL character (for instance, '^D' is a CTRL-D); '^?' is interpreted as DEL and '^–' is interpreted as undefined.

**min** *i* Set the **MIN** value to *i*.

**time** *i* Set the **TIME** value to *i*.

**rows** *n* Set the recorded number of rows on the terminal to *i*.

**columns** *i* Set the recorded number of columns on the terminal to *i*.

**cols** *i* An alias for **columns** *i*.

**Combination Modes**

**cooked** Process the **ERASE, WERASE, KILL, INTR, QUIT, EOF, EOL, EOL2, STOP, START, SUSP, RPRNT, FLUSH,** and **LNEXT,** characters specially, and perform output post-processing.

**evenp** or **parity**
Enable **parenb** and **cs7.**

**oddp** Enable **parenb, cs7,** and **parodd.**

**–parity, –evenp,** or **–oddp**
Disable **parenb,** and set **cs8.**

[–]**raw** Enable raw input and output. With a '–', disable raw I/O. In raw mode, there is no special processing of the **ERASE, WERASE, KILL, INTR, QUIT, EOF, EOL, EOL2, STOP, START, SUSP, RPRNT, FLUSH,** nor **LNEXT** characters, nor is there any output post-processing.

**[–]nl** Unset **icrnl, onlcr**. With a '–', set them. In addition **–nl** unsets **inlcr, igncr, ocrnl,** and **onlret**.

**[–]lcase** Set **xcase, iuclc,** and **olcuc**. With a '–', unset them.

**[–]LCASE** Same as **lcase (–lcase)**.

**[–]tabs**
**tab3** Preserve TAB characters when printing. With a '–', or with **tab3**, expand TAB characters to SPACE characters.

**ek** Reset the **ERASE** and **KILL** characters back to normal: **DEL** and CTRL-**U**).

**sane** Reset all modes to some reasonable values.

**crt** Set options for a CRT (**echoe, echoctl,** and, if >= 1200 baud, **echoke**.)

**dec** Set all modes suitable for Digital Equipment Corp. operating systems users (**ERASE, KILL,** and **INTR** characters to **^?, ^U,** and **^C, decctlq,** and **crt**.)

**term** Set all modes suitable for the terminal type **term**, where **term** is one of **tty33, tty37, vt05, tn300, ti700,** or **tek**.

**SEE ALSO**

**ioctl**(2), **termio**(4)

## NAME

stty_from_defaults – set terminal editing characters from the defaults database

## SYNOPSIS

**stty_from_defaults**

## AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**stty_from_defaults** is a utility provided with the *SunView* environment.

**stty_from_defaults** sets the three editing characters (to erase a character, erase a word, and kill a line) according to the choices in your defaults database. It does not set any other tty options. If you run **stty**(1V) in your **.login** or **rc.local** files, you may want to run **stty_from_defaults** immediately after it. This will override any settings of *stty*erase, *stty*werase, or *stty*kill, so that you will have the same character-editing behavior with SunView application programs.

To specify the editing characters **stty_from_defaults** will set, run **defaultsedit**(1) and select the "Text" category. The editing characters are called Edit_back_char, Edit_back_word, and Edit_back_line. Type the value you want to the right of each item. To specify CTRL-X, type '**X**'— that is, the three characters '\', '^', and '**X**'. To specify DEL, type '**?**'.

If you do not specify your own values, the default values are DEL, CTRL-**W**, and CTRL-**U**, respectively.

## SEE ALSO

**defaultsedit**(1), **stty**(1V), **sunview**(1)

*SunView 1 Beginner's Guide*

## NAME

su – super-user, temporarily switch to a new user ID

## SYNOPSIS

**su** [ – ] [ **–f** ] [ **–c** *command* ] [ *username* ]

## DESCRIPTION

**su** creates a new shell process that has the user ID for the specified *username* as its real and effective user ID. **su** asks for the password, just as if you were logging in as *username*, and, if the password is given, changes the real and effective user IDs and group IDs and group set to those of *username* and invokes the shell specified in the password file for that *username*, without changing the current directory. The user environment is thus unchanged except for **HOME** and **SHELL**, which are taken from the password file for the user being substituted (see **environ**(5V)). If *username* is **root**, **USER** is changed to **root**. The new user ID stays in force until the shell exits.

The new shell will not be a login shell, so it will not read *username*'s **.login** or **.profile** files, but it will read any other configuration files for that user (for instance, the **.cshrc** file for the C shell) just as if that user had invoked a new shell.

If no *username* is specified, **root** is assumed. If the **wheel** group (group 0) has members, only they can **su** to **root**, even with the root password. To remind the super-user of his responsibilities, the shell substitutes '#' for '$' or '%' in its usual prompt.

## OPTIONS

– Perform a complete login. Remove all variables from the environment except for **TERM**, set **USER** to *username*, set **HOME** and **SHELL** as specified above, set **PATH** to **:/usr/ucb:/bin:/usr/bin**, change directories to *username*'s home directory, and tell the shell to read *username*'s **.login** or **.profile** file.

**–f** Perform a fast **su** by passing the **–f** flag to the shell. This flag is for use with the C shell; it will prevent the C shell from reading *username*'s **.cshrc** file.

**–c** *command*
Execute *command* after logging in as the new user.

## FILES

**.cshrc**
**.login**
**.profile**

## SEE ALSO

**csh**(1), **sh**(1), **environ**(5V)

## BUGS

**su** fails when run from within a subdirectory of a directory that *username* either cannot search, or cannot read (that is, *username* does not have both read and execute permission).

**su** fails to reset the user ID to root when the current working directory is in an NFS-mounted file system, and does not have its search permission set for "other" users.

NAME

sum – calculate a checksum for a file

SYNOPSIS

**sum** *filename*

SYSTEM V SYNOPSIS

**/usr/5bin/sum** [ **–r** ] *filename*

DESCRIPTION

**sum** calculates and displays a 16-bit checksum for the named file, and also displays the size of the file in kilobytes. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The checksum is calculated by an algorithm which may yield different results on machines with 16-bit **int**s and machines with 32-bit **int**s, so it cannot always be used to validate that a file has been transferred between machines with different-sized **int**s.

SYSTEM V DESCRIPTION

**sum** calculates and prints a 16-bit checksum for the named file, and also prints the number of 512-byte blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. This algorithm is independent of the size of **int**s on the machine.

SYSTEM V OPTIONS

The option **–r** causes the (machine-dependent) algorithm used by the non-System V **sum** to be used in computing the checksum.

SEE ALSO

**wc**(1)

DIAGNOSTICS

**Read error** is indistinguishable from EOF on most devices; check the block count.

NAME

sunview – the SunView window environment

SYNOPSIS

**sunview** [ **–i** ] [ **–p** ] [ **–B** | **–F** | **–P** ] [ **–S** ] [ **–8bit_color_only** ] [ **–overlay_only** ] [ **–toggle_enable** ]
[ **–b** *red green blue* ] [ **–d** *display-device* ] [ **–f** *red green blue* ] [ **–k** *keyboard-device* ]
[ **–m** *mouse-device* ] [ **–n**|**–s** *startup-filename* ] [ **–background** *raster-filename* ]
[ **–pattern on** | **off** | **gray** | *iconedit-filename* ]

DESCRIPTION

**sunview** starts up the SunView environment and (unless you have specified otherwise) a default layout of a few useful "tools," or window-based applications.

See **Start-up Processing** below to learn how to specify your own initial layout of tools. Some of the behavior of **sunview** is controlled by settings in your defaults database; see **SunView Defaults** below, and **defaultsedit**(1) for more information.

To exit **sunview** use the **Exit SunView** menu item. In an emergency, type CTRL-D then CTRL-Q (there is no confirmation in this case).

OPTIONS

**–i** Invert the background and foreground colors used on the screen. On a monochrome monitor, this option provides a video reversed image. On a color monitor, colors that are not used as the background and foreground are not affected.

**–p** Print to the standard output the name of the window device used for the **sunview** background.

**–B** Use the "background color" ( **–b** ) for the background.

**–F** Use the "foreground color" ( **–f** ) for the background.

**–P** Use a stipple pattern for the background. This option is assumed unless **–F** or **–B** is specified.

**–S** Set **Click-to-type** mode, allowing you to select a window by clicking in it. Having done so, input is directed to that window regardless of the position of the pointer, until you click to select some other window.

**–8bit_color_only**
For multiple plane group frame buffers, only let windows be created in the 8 bit color plane group. This frees up the black and white overlay plane to have a separate desktop running on it. This option is usually used with the **–toggle_enable** option . See **Multiple Desktops on the Same Screen**, below.

**–overlay_only**
For multiple plane group frame buffers, only let windows be created in the black and white overlay plane group. This frees up the 8 bit color plane group to have a separate desktop running in it. This option is usually used with the **–toggle_enable** option. See **Multiple Desktops on the Same Screen**, below.

**–toggle_enable**
For multiple plane group frame buffers, when sliding the pointer between different desktops running within different plane groups on the same screen, change the enable plane to allow viewing of the destination desktop. See **Multiple Desktops on the Same Screen**, below.

**–b** *red green blue*
Specify values for the *red*, *green* and *blue* components of the background color. If this option is not specified, each component of the background color is 255 (white). Sun 3/110 system users that use this option should use the **–8bit_color_only** option as well.

**–d** *display-device*
Use *display-device* as the output device, rather than **/dev/fb** the default frame buffer device.

**–f** *red green blue*
Specify values for the *red*, *green* and *blue* components of the foreground color. If this option is not specified, each component of the foreground color is 0 (black). Sun 3/110 system users that use this option should use the **–8bit_color_only** option as well.

**–k** *keyboard-device*
Accept keyboard input from *keyboard-device*, rather than **/dev/kbd**, the default keyboard device.

**–m** *mouse-device*
Use *mouse-device* as the system pointing device (locator), rather than **/dev/mouse**, the default mouse device.

**–n** Bypass startup processing by ignoring the **/usr/lib/sunview** and **/sunview** (and **/.suntools**) files.

**–s** *startup-filename*
Read startup commands from *startup-filename* instead of **/usr/lib/sunview** or **/.sunview**).

**–background** *raster-filename*
Use the indicated raster file as the image in your background. The raster file can be created with **screendump**(1). Screen dumps produced on color monitors currently do not work as input to this option. Small images are centered on the screen.

**–pattern on | off | gray |** *iconedit-filename*
Use the indicated "pattern" to cover the background. **on** means to use the default desktop gray pattern. **off** means to not use the default desktop gray pattern. **gray** means to use a 50% gray color on color monitors. *iconedit-filename* is the name of a file produced with **iconedit**(1) which contains an image that is to be replicated over the background.

## USAGE

### Windows

The SunView environment always has one window open, referred to as the background, which covers the whole screen. A solid color or pattern is its only content. Each application is given its own window which lies on top of some of the background (and possibly on top of other applications). A window obscures any part of another window which lies below it.

### Input to Windows

Mouse input is always directed to the window that the pointer is in at the time. Keyboard input can follow mouse input or, it can remain within a designated window using the **Click-to-Type** default setting. If you are not using **Click-to-Type**, and the pointer is on the background, keyboard input is discarded. Input actions (mouse motions, button clicks, and keystrokes) are synchronized, which means that you can "type-ahead" and "mouse-ahead," even across windows.

### Mouse Buttons

LEFT mouse button
Click to select or choose objects.

MIDDLE mouse button
In text, click once to shorten or lengthen your selection. In graphic applications or on the desktop, press and hold to move objects.

RIGHT mouse button
Press and hold down to invoke menus.

### Menus

**sunview** provides pop-up menus. There are two styles of pop-up menus: an early style, called "stacking menus," and a newer style, called "walking menus" (also known as "pull-right menus"). In the current release, walking menus are the default; stacking menus are still available as a defaults option.

Usually, a menu is invoked by pressing and holding the RIGHT mouse button. The menu remains on the screen as long as you hold the RIGHT mouse button down. To choose a menu item, move the pointer onto it (it is then highlighted), then release the RIGHT mouse button.

Another available option is "stay-up menus." A stay-up menu is invoked by pressing and releasing the RIGHT mouse button. The menu appears on the screen after you release the RIGHT mouse button. To choose a menu item, move the pointer onto it (it is then highlighted), then press and release the RIGHT mouse button a second time. Stay-up menus are an option in your defaults database; see **SunView Defaults** below.

With walking menus, any menu item can have an arrow pointing (⇒) to the right. Moving the pointer onto this arrow pops up a "sub-menu," with additional items. Choosing the item with an arrow (the "pull-right item") invokes the first item on the sub-menu.

### The SunView Menu

You can use the default SunView menu to start SunView applications and perform some useful functions. To invoke it, hold down the RIGHT mouse button when the pointer is anywhere in the background.

The default SunView menu consists of four sub-menus, labeled **Shells, Editors, Tools,** and **Services,** along with an **Exit** SunView item. These sub-menus contain the following items:

**Shells**

**Command Tool** Bring up a **cmdtool**(1), a scrollable window-based terminal emulator that supports a shell.

**Shell Tool** Bring up a **shelltool**(1), an tty-based terminal emulator that supports a shell.

**Graphics Tool** Bring up a **gfxtool**(1), for running graphics programs.

**Console** Bring up a Console window, a **cmdtool** with the –C flag, to act as the system console. Since many system messages can be directed to the console, there should always be a console window on the screen.

**Editors**

**Text Editor** Bring up a **textedit**(1), for reading and editing text files.

**Defaults Editor** Bring up a **defaultsedit**(1), for browsing or changing your defaults settings.

**Icon Editor** Bring up a new **iconedit**(1).

**Font Editor** Bring up a **fontedit**(1).

**Tools**

**Mail Tool** Bring up a **mailtool**(1), for reading and sending mail.

**Dbx (Debug) Tool** Bring up a **dbxtool**(1), a window-based source debugger.

**Performance Meter** Bring up a **perfmeter**(1) to monitor system performance.

**Clock** Bring up a new **clock**(1).

**Services**

**Redisplay All** Redraw the entire screen. Use this to repair damage done by processes that wrote to the screen without consulting the SunView system.

**Printing** There are two items on this submenu, **Check Printer Queue** and **Print Selected Text**. **Check Printer Queue** displays the printer queue in your console; **Print Selected Text** sends selected text to the standard printer.

**Remote Login** There are two items on this submenu, **'Command Tool'** and **'Shell Tool'**. Each creates a terminal emulator that prompts for a machine name and then starts a shell on that machine.

**Save Layout** Writes out a **/.sunview** file that **sunview** can then use when starting up again. An existing **/.sunview** file is saved as **/.sunview–**.

**Lock Screen** Completely covers the screen with a graphics display, and "locks" the workstation until you type your password. When you "unlock" the workstation, the screen is restored as it was when you locked it. See **lockscreen**(1) for details.

**Exit SunView**

Exit from **sunview**, including all windows, and kill processes associated with them. You return to the shell from which you started **sunview**.

You can specify your own SunView menu; see **SunView Defaults** below for details.

**The Frame Menu**

A small set of universal functions are available through the Frame menu. There are also accelerators for some of these functions, described under **Frame Menu Accelerators**, below.

You can invoke the Frame menu when the cursor is over a part of the application that does not provide an application-specific menu, such as the frame header (broad stripe holding the application's name), the border stripes of the window, and the icon.

**Close**

**Open** Toggle the application between closed (iconic) and open state. Icons are placed on the screen according to the icon policy in your defaults database; see **SunView Defaults** below. When a window is closed, its underlying processes continue to run.

**Move** Moves the application window to another spot on the screen. **Move** has a sub-menu with two items: **Unconstrained** and **Constrained**.

- **Unconstrained** Move the window both horizontally and vertically.
- **Constrained** Moves are either vertical or horizontal, but not both.

  Choosing **Move** invokes an **Unconstrained** move.

**Resize** Shrink or stretch the size of a window on the screen. **Resize** has a sub-menu containing:

- **Unconstrained** Resize the window both horizontally and vertically.
- **Constrained** Resize vertically or horizontally, but not both.

  Choosing **Resize** invokes an **Unconstrained** resize.
- **UnZoom**
- **Zoom** **Zoom** expands a window vertically to the full height of the screen. **UnZoom** undoes this.
- **FullScreen** Make a window the full height and width of the screen.

**Front** Bring the window to "the top of the pile." The whole window becomes visible, and hides any window it happens to overlap on the screen.

**Back** Put the window on the "bottom of the pile". The window is hidden by any window which overlaps it.

**Props** Display the property sheet. (Only active for applications that provide a property sheet.)

**Redisplay** Redraw the contents of the window.

**Quit** Notify the application to terminate gracefully. Requires confirmation.

**Frame Menu Accelerators**

Accelerators are provided for some Frame menu functions. You can invoke these functions by pushing a single button in the window's frame header or outer border. See the **SunView Beginner's Guide** for more details.

**Open** Click the LEFT mouse button when the pointer is over the icon.

**Move** Press and hold the MIDDLE mouse button while the pointer is in the frame header or outer border. A bounding box that tracks the mouse is displayed while you hold the button down. When you release the button, the window is redisplayed within the bounding box. If the pointer is near a corner, the move is **Unconstrained**. If it is in the center third of an edge, the move is **Constrained**.

**Resize** Hold the CTRL key and press and hold the MIDDLE mouse button while the pointer is in the frame header or outer border. A bounding box is displayed, and one side or corner tracks the mouse. If the pointer is near a corner when you press the mouse button, the resize is **Unconstrained**; if in the middle third of an edge, the resize is **Constrained**.

**Zoom**
**UnZoom** Hold the CTRL key and click the LEFT mouse button while the pointer is in the frame header or outer border.

**Front** Click the LEFT mouse button while the pointer is on the frame header or outer border.

**Back** Hold the SHIFT key and click the LEFT mouse button while the pointer is on the frame header or outer border.

In addition, you can use two function keys as even faster accelerators. To expose a window that is partially hidden, press the **Front** function key (normally L5) while the pointer is anywhere in that window. Or, if the window is completely exposed, use the **Front** key to hide it. Similarly, to close an open window, press the **Open** key (normally L7) while the pointer is anywhere in that window. If the window is iconic, use the **Open** key to open it.

In applications with multiple windows, you can often adjust the border between two windows up or down, without changing the overall size of the application: hold the CTRL key, press the MIDDLE mouse button over the boundary between the two windows, and adjust the size of the (bounded) subwindow as with **Resize**.

**Startup Processing: The .sunview File**

Unless you override it, **sunview** starts up with a predefined layout of windows. The default layout is specified in the file **/usr/lib/sunview**. If there is a file called **.sunview** in your home directory, it is used instead. For compatibility with earlier releases, if there is no **.sunview** file in your home directory, but a **.suntools** file instead, the latter file is used.

**SunView Defaults**

SunView allows you to customize the behavior of applications and packages by setting options in a defaults database (one for each user). Use **defaultsedit**(1) to browse and edit your defaults database. Select the "SunView" category to see the following items (and some others):

**Walking_menus** If enabled, the SunView menu, the Frame menu, and many applications will use walking menus. Applications that have not been converted will still use stacking menus. If disabled, applications will use stacking menus. The default value is "Enabled."

**Click_to_Type** If enabled, keyboard input will stay in a window until you click the LEFT or MIDDLE mouse button in another window. If disabled, keyboard input will follow the mouse. The default value is "Disabled."

**Font** You can change the SunView default font by giving the full pathname of the font you want to use. Some alternate fonts are in the directory **/usr/lib/fonts/fixedwidthfonts**. The default font from the SunOS 2.0 release was **/usr/lib/fonts/fixedwidthfonts/screen.r.13**. The default value is null, which has the same effect as specifying **/usr/lib/fonts/fixedwidthfonts/screen.r.11**.

**Rootmenu_filename** You can change the SunView menu by giving the full pathname of a file that specifies your own menu. See **The SunView Menu File** below for details. The default value is null, which gives you the menu found in **/usr/lib/rootmenu**.

**Icon_gravity** Determine which edge of the screen ("North", "South", "East", or "West") icons will place themselves against. The default value is "North."

**Audible_bell** If enabled, the "bell" command will produce a beep. The default value is "Enabled.

**Visible_bell** If enabled, the "bell" command will cause the screen to flash. The default value is "Enabled."

**Root_Pattern** Used to specify the "pattern" that covers the background. "on" means to use the default desktop gray pattern. "off" means to not use the default desktop gray pattern. "gray" means to use a 50% gray color on color monitors. Anything else is the name of a file produced with **iconedit**(1) which contains an image that is replicated all over the background. The default value is "on."

After you have set the options you want in the "SunView" category, click on the **Save** button in **defaultsedit**; then exit **sunview** and restart it.

Select the "Menu" category to see the following items (and some others):

**Stay_up** If enabled, menus are invoked by pressing and releasing the RIGHT mouse button; the menu appears after you release the RIGHT mouse button. To choose a menu item, point at it, then press and release the RIGHT mouse button a second time. The default value is "False".

**Items_in_column_major**
If enabled, menus that have more than one column are presented in "column major" order (the way **ls**(1) presents file names). This may make a large menu easier to read. The default value is "False."

After you have set the options you want in the "Menu" category, click on the **Save** button in **defaultsedit**. Any applications you start after saving your changes will be affected by your new choices. For all defaults categories except for "SunView", you do *not* need to exit **sunview** and restart it.

**The SunView Menu File**

The file called **/usr/lib/rootmenu** contains the specification of the default SunView menu. You can change the SunView menu by creating your own file and giving its name in the **Rootmenu_filename** item in the SunView Defaults.

Lines in the file have the following format: The left side is a menu item to be displayed, and the right side is a command to be executed when that menu item is chosen. You can also include comment lines (beginning with a '#') and blank lines.

The menu item can be a string, or the full pathname of an icon file delimited by angle brackets (unless **Walking_menus** is disabled in the SunView defaults). Strings with embedded blanks must be delimited by double quotes.

There are four reserved-word commands that can appear on the right side.

**EXIT** Exit **sunview** (requires confirmation).

**REFRESH** Redraw the entire screen.

**MENU** This menu item is a pull-right item with a submenu. If a full pathname follows the **MENU** command, the submenu contents are taken from that file. Otherwise, all the lines

between a **MENU** command and a matching **END** command are added to the submenu.

**END** Mark the end of a nested submenu. The left side of this line should match the left side of a line with a **MENU** command.

If the command is not one of these four reserved-word commands, it is treated as a command line and executed. No shell interpretation is done, although you can run a shell as a command.

Here is a menu file that demonstrates some of these features:

```
Quit                    EXIT
Mail reader             mailtool
My tools                MENU  /home/me/mytools.menu
Click to type           swin -c
Follow mouse            swin -m
Print selection         sh -c get_selection | lpr
Nested menu             MENU
    Command Tool        cmdtool
    Shell Tool          shelltool
Nested menu             END
Icon menu               MENU
    </usr/include/images/textedit.icon>     textedit
    </usr/include/images/dbxtool.icon>      dbxtool
Icon menu               END
```

**Multiple Screens**

The **sunview** program runs on either a monochrome or color screen. Each screen on a machine with multiple screens may have a separate **sunview** running. The keyboard and mouse input devices can be shared between screens. Using **adjacentscreens**(1) you can set up the pointer to slide from one screen to another when you move it off the edge of a screen.

To set up an instance of **sunview** on two screens:

1. Invoke **sunview** on the first display as you normally would. This starts an instance of **sunview** on the default frame buffer (**/dev/fb**).
2. In a **shelltool**, run '**sunview -d** *device* &'. This starts another device. (A typically choice might be **/dev/cgone**).
3. In that same **shelltool**, run '**adjacentscreens /dev/fb -r** *device*'. This sets up the cursor to switch between screens as it crosses the right or left edge of the respective screens.

**Multiple Desktops on the Same Screen**

Machines that support multiple plane groups, such as the Sun-3/110 system can support independent **sunview** processes on each plane group. They can share keyboard and mouse input in a manner similar to that for multiple screens. To set up two plane groups:

1. Invoke **sunview** in the color plane group by running '**sunview -8bit_color_only -toggle_enable**'. This starts **sunview** on the default frame buffer named **/dev/fb**, but limits access to the color plane group.
2. In a **shelltool**, run '**sunview -d /dev/bwtwo -toggle_enable -n &**.' This starts **sunview** in the overlay plane accessed by **/dev/bwtwo**.
3. Run '**adjacentscreens -c /dev/fb -l /dev/bwtwo**'. This sets up the pointer to switch between desktops as it crosses the right or left edge of the respective desktops.

Pre-3.2 applications cannot be run on the **–8bit_color_only** desktop, because they do not write to the overlay plane.

**switcher**(1), another application for switching between desktops, uses some amusing video wipe animation. It can also be used to toggle the enable plane. See **switcher**(1) for details.

**Generic Tool Arguments**

Most window-based tools take the following arguments in their command lines:

| FLAG | (LONG FLAG) | ARGUMENTS | NOTES |
|---|---|---|---|
| **–Ww** | **(–width)** | **columns** | |
| **–Wh** | **(–height)** | **lines** | |
| **–Ws** | **(–size)** | **x y** | **x and y are in pixels** |
| **–Wp** | **(–position)** | **x y** | **x and y are in pixels** |
| **–WP** | **(–icon_position)** | **x y** | **x and y are in pixels** |
| **–Wl** | **(–label)** | **"string"** | |
| **–Wi** | **(–iconic)** | | **makes the application start iconic (closed)** |
| **–Wt** | **(–font)** | **filename** | |
| **–Wn** | **(–no_name_stripe)** | | |
| **–Wf** | **(–foreground_color)** | **red green blue** | **0-255 (no color-full color)** |
| **–Wb** | **(–background_color)** | **red green blue** | **0-255 (no color-full color)** |
| **–Wg** | **(–set_default_color)** | | **(apply color to subwindows too)** |
| **–WI** | **(–icon_image)** | **filename** | **(for applications with non-default icons)** |
| **–WL** | **(–icon_label)** | **"string"** | **(for applications with non-default icons)** |
| **–WT** | **(–icon_font)** | **filename** | **(for applications with non-default icons)** |
| **–WH** | **(–help)** | | **print this table** |

Each flag option may be specified in either its short form or its long form; the two are completely synonymous.

**SunView Applications**

Some of the applications that run in the SunView environment:

**clock**(1), **cmdtool**(1), **dbxtool**(1), **defaultsedit**(1), **fontedit**(1), **gfxtool**(1), **iconedit**(1), **lockscreen**(1), **mailtool**(1), **overview**(1), **perfmeter**(1), **shelltool**(1), **tektool**(1), **textedit**(1), **traffic**(1)

Some of the utility programs that run in or with the SunView environment:

**adjacentscreens**(1), **clear_functions**(1), **get_selection**(1), **stty_from_defaults**(1), **swin**(1), **switcher**(1), **toolplaces**(1)

## ENVIRONMENT

**DEFAULTS_FILE**

The value of this environment variable indicates the file from which SunView defaults are read. When it is undefined, defaults are read from the **.defaults** file in your home directory.

## FILES

**/.sunview**
**/usr/bin/sunview**
**/usr/lib/rootmenu**
**/usr/lib/fonts/fixedwidthfonts/***
**/dev/win***x***
**/dev/pty***p***x**
**/dev/tty***p***x**
**/dev/fb**
**/dev/kbd**

**/dev/mouse**
**/etc/utmp**

SEE ALSO

**adjacentscreens**(1), **clear_functions**(1), **clock**(1), **cmdtool**(1), **dbxtool**(1), **defaultsedit**(1), **fontedit**(1), **get_selection**(1), **gfxtool**(1), **iconedit**(1), **lockscreen**(1), **mailtool**(1), **overview**(1), **perfmeter**(1), **screendump**(1), **shelltool**(1), **stty_from_defaults**(1), **swin**(1), **switcher**(1), **tektool**(1), **textedit**(1), **toolplaces**(1), **traffic**(1)

BUGS

Console messages ignore window boundaries unless redirected to a console window. This can disrupt the **sunview** desktop display. The display can be restored using the **Redisplay All** item on the SunView menu. To prevent this, use the **Console** item to start a console window.

With an optical mouse, sometimes the arrow-shaped cursor does not move at start-up; moving the mouse in large circles on its pad normally brings it to life.

**sunview** requires that the **/etc/utmp** file be given read and write permission for all users.

On a color display, colors may "go strange" when the cursor is in certain windows that request a large number of colors.

When running multiple desktops, only one console window can be used.

In **Click-to-type** mode, it is impossible to exit from **sunview** by typing CTRL-D CTRL-Q.

NAME

swin – set or get SunView user input options

SYNOPSIS

**swin** [ **–cghm** ] [ **–r** *event value shift_state* ] [ **–s** *event value shift_state* ] [ **–t** *seconds* ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

The **swin** (set window; analogous to **stty**(1V)) command lets you change some of the input behavior of your SunView environment. By default, your keyboard input follows your pointer. This means that in order to type to a window you position the pointer over the window. This is called *keyboard-follows-mouse* mode.

You can specify that the keyboard input continues to go to the same window, regardless of the pointer position, until you take some specific action, like clicking the mouse. When this is done, you can roam around the screen with the pointer and not change the window to which keyboard input is directed. Running SunView like this is said to be operating in *click-to-type* mode.

When running in click-to-type mode, one user action *sets* the type-in point in the window that you want to receive keyboard input. The default user action to do this is the clicking of the LEFT mouse button while positioning the pointer over the new type-in point. This user action can be changed.

Another user action *restores* the previous type-in point in the window that you want to receive keyboard input. The default user action to do this is the clicking of the MIDDLE mouse button while positioning the pointer over the window. This user action can be changed.

OPTIONS

**–c** Turn on click-to-type mode using the default user actions: the LEFT mouse button sets the type-in point and the MIDDLE mouse button restores the type-in point. You can use the **defaultsedit**(1) program to set click-to-type on permanently; see the **Click_to_Type** option of **sunview**(1).

**–g** Get the state of the user input options controlled by **swin**. If no arguments are supplied to **swin** then **–g** is implied.

**–h** Print out a help message that briefly describes the options to **swin**.

**–m** Run in keyboard-follows-mouse mode.

**–s** *event value shift_state*

Set the user action that sets the type-in point and sets the keyboard input window. The *event* identifies the particular user action and is one of:

LOC_WINENTER
: pointer entering a window

MS_LEFT
: LEFT mouse button

MS_MIDDLE
: MIDDLE mouse button

MS_RIGHT
: RIGHT mouse button

*decimal_number*
: place the decimal number of a firm event here; see list of events in **/usr/include/sundev/vuid_event.h** (avoid function keys, normally unused control-ascii characters are OK, normally unused SHIFT keys are OK).

*value* identifies the transition of the *event* and is one of:

ENTER the pointer entering a window (use with LOC_WINENTER)

DOWN the button associated with *event* went down

UP the button associated with *event* went up (avoid this)

The *shift_state* identifies the state of the SHIFT keys at the time of the *event/value* pair in order for that pair to be used to control the keyboard input window. The *shift_state* is one of:

SHIFT_DONT_CARE
Ignore the state of the SHIFT keys

SHIFT_ALL_UP
All the SHIFT keys must be up

SHIFT_LEFT
The left SHIFT key must be down (not the key labeled LEFT)

SHIFT_RIGHT
the right SHIFT key must be down (not the key labeled RIGHT)

SHIFT_LEFTCTRL
the left CTRL key must be down

SHIFT_RIGHTCTRL
the right CTRL key must be down

**–r** *event value shift_state*
Set the user action that restores the type-in point and sets the keyboard input window. This user action is swallowed so that the application that owns the window does not see it. However, if the window already has keyboard input or if the window refuses keyboard input then this user action is passed on through to the application. The parameters to this command are like those for **–s**. The following example shows modifying the default click-to-type user actions so that a SHIFT left is required for the restore user event:

**example% swin -c -r MS_MIDDLE DOWN SHIFT_LEFT**

**–t** *seconds*
SunView synchronizes input so that it does not hand out the next user action until the application fielding the current user action finishes its processing. This allows type-ahead and mouse-ahead. If an application does not finish processing within a given length of time (process virtual time; not wall clock time), the next user action is handed out anyway. This avoids any one application from hanging the workstation. The **–t** command sets this time limit. A *seconds* value of 0 tells SunView to run unsynchronized; beware of race conditions in this mode. The default seconds value is 2 and the **–c** command makes it 10 seconds.

FILES

**/usr/include/sundev/vuid_event.h**
list of event codes

SEE ALSO

**defaultsedit**(1), **stty**(1V), **sunview**(1)

*SunView 1 Beginner's Guide*

DIAGNOSTICS

**swin not passed parent window in environment**
**swin** does not work unless SunView is started already.

**BUGS**

**swin** gets you no help in preventing you from specifying **–r** or **–s** parameters that are not sensible.

NAME

switcher – switch attention between multiple SunView desktops on the same physical screen

SYNOPSIS

**switcher** [ **–d** *frame-buffer* ] [ **–s n | l | r | i | o | f** ] [ **–m** *x y* ] [ **–n** ] [ **–e 0 | 1** ]

AVAILABILITY

This command is available for Sun-2, Sun-3 and Sun-4 systems with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**switcher** is used as an alternative to **adjacentscreens**(1) for getting between desktops on the Sun-3/110. Clicking the switcher icon gets you to another desktop using some amusing video-wipe animation. When using walking menus, a menu is available to invoke the switch as well. **switcher** can also be used to simply set the enable plane to 0 or 1 should it get out of wack.

OPTIONS

**–d** *frame-buffer*

The *frame buffer* is a frame buffer device name, such as **/dev/fb**, **/dev/cgfour** or **/dev/bwtwo**, on which the desktop that you want to get to resides. This name is the same one supplied to **sunview** The **–d** flag is optional; if not specified, the default device is **/dev/fb**

**–s n | l | r | i | o | f**

The **–s** flag specifies the type of animation used when switching: **n** (now), **l** (left wipe), **r** (right wipe), **i** (tunnel in), **o** (tunnel out), or **f** (fade). The **–s** flag is optional because if not specified, the default animation is to switch immediately. **n** (now) mode.

**–m***x y* The **–m** indicates what the mouse position should be on the destination desktop after the switch. An (*x y*) value-pair of (–1 –1) says to use the position of the mouse on the desktop at the time of the switch as the mouse position on the destination desktop. The **–m** flag is optional; if not specified, the default is (–1 –1).

**–n** The **–n** flag means no switcher icon is wanted so do the switch right now and exit **switcher** after the switch. This is handy if you want to switch from a root menu command.

**–e 0 | 1** The **–e** flag causes the overlay enable plane of the device specified with the **-d** flag to be set to either 0 (show color) or 1 (show black and white). **switcher** run with this option has nothing to do with SunView, only the enable plane is set.

EXAMPLE

A common multiple desktop configuration for the Sun-s/110 is one monochrome and one color desktop. You could set up an instance of **sunview**(1) on each plane group in the following way:

1. Invoke **sunview** in the color plane group by running:

   **example% sunview –8bit_color_only –toggle_enable**

   This starts **sunview** on the default frame buffer named **/dev/fb** but limits access to the color plane group.

2. In a **shelltool**(1), run:

   **example% sunview –d /dev/bwtwo –toggle_enable &**

   This starts **sunview** in the overlay plane that is accessed by **/dev/bwtwo**.

3. In a **shelltool** on the original desktop run:

   **example% switcher –d /dev/bwtwo –s i &**

   Clicking on the switcher icon when it is visible moves you to the **/dev/bwtwo** desktop.

4. In a **shelltool** on the **/dev/bwtwo** desktop run:

   **example% switcher –s o &**

Clicking on the switcher icon when it is visible moves you back to the **/dev/fb** desktop.

FILES

**/usr/bin/switcher**
**/dev/bwtwo**
**/dev/fb**
**/dev/cgfour**

SEE ALSO

**adjacentscreens**(1), **shelltool**(1), **sunview**(1)

NAME

symorder – rearrange a list of symbols

SYNOPSIS

**symorder** *orderlist symbolfile*

DESCRIPTION

*orderlist* is a file containing symbols to be found in symbolfile, 1 symbol per line.

*symbolfile* is updated in place to put the requested symbols first in the symbol table, in the order specified. This is done by swapping the old symbols in the required spots with the new ones. If all of the order symbols are not found, an error is generated.

This program was specifically designed to cut down on the overhead of getting symbols from **/vmunix**.

Sun386i DESCRIPTION

Symbols specified on the command line are moved to the beginning of the output symbol table, not swapped. Therefore, the symbols specified on the command line will appear in order at the beginning of the output symbol table, followed by the original symbol table with the gaps created by the moved symbols closed.

FILES

**/vmunix**

SEE ALSO

**nlist**(3)

NAME

sync – update the super block; force changed blocks to the disk

SYNOPSIS

**sync**

DESCRIPTION

**sync** forces any information on its way to the disk to be written out immediately. **sync** can be called to ensure that all disk writes are completed before the processor is halted abnormally.

SEE ALSO

**cron**(8), **fsck**(8), **halt**(8), **reboot**(8)

## NAME

sysex – invoke the system exerciser

## SYNOPSIS

**sysex**

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

**sysex** is the system exerciser for Sun386i systems.

This program is designed to run under the SunView environment, but a dumb terminal interface is provided. The program tests subsystems of a Sun386i system, printing information to the console and log files. Most commands are accessible by way of buttons, toggle switches, or menus. A startup file **.sysexrc**, can be created by experienced users to set runtime parameters.

## USAGE

### Subwindows

| | |
|---|---|
| Control Panel | Tells which version of the exerciser is running. User controls **sysex** executions through the toggles, buttons, and sliders in this panel. |
| Perfmon window | Graphically displays system statics. The standard SunView **perfmeter**(1). |
| Console window | Displays the system messages and **sysex** error messages. |
| Status Window | Displays pass counts and error counts for all tests that are currently selected. Also displays system pass count, and total system errors. Elapsed time signifies time since start button is pressed. |

### Load Sliders

These slide bars allow the user to modulate the load on the system.

I/O-CPU Load
: Moving this slider changes the balance between I/O-intensive and compute-intensive tests that are running.

SYSTEM LOAD
: Moving this slider increases and decreases the system activity generated by **sysex.**

### Test Toggles

Each test selection on the control panel is selectable by moving the cursor over to the toggle and pressing the left mouse button.

The tests are displayed by device groups in the control panel. A test is enabled when a check mark is seen in the box. Clicking left on the group label acts as a group enable/disable for all Wed tests in that device group. Currently there are tests for physical memory and virtual memory, fixed disk, diskette, Ethernet, and color frame buffer.

### Command Buttons

Command buttons exist for the following commands:

| | |
|---|---|
| **Quit Sysex** | Stop all current tests and exit the exerciser. All logs will be saved. |
| **Log Files** | Display menu for choosing a log to view, reset, or print. |
| **Options** | Display window through which **.sysexrc** parameters can be modified. |
| **Print Screen** | Take screendump of the current screen. |
| **Start Tests** | Start all test from pass0 that have been selected. Resets pass count. Toggles to Stop Tests. Begin elapsed time count. |
| **Stop Tests** | Stop all tests that are running. Toggles to Start Tests. |
| **Pause** | Pause tests by issuing SIGSTP. |

**Continue** Continue testing from the stopped state without resetting pass count. Toggles to Pause,leaves pass counts intact. Will continue elapsed time count if Continued from a Pause.

**Logs**

When the user selects the DISPLAY LOGS button and chooses from the log menu, a scrollable pop-up window will display the log, which can be one of **/var/sysex/sysex.info**, **/var/sysex/sysex.error**, or **/etc/adm/messages**. Logs contain messages classified as INFO, WARNING, ERROR, or FATAL. The INFO file contains all messages; the ERROR file contains only error and fatal messages.

**Variables**

The **sysex** program has several variables than can be set in the **.sysexrc** file. Some of the variables pertain to only one test and others are global to all tests. Clicking Done will save changes to the **.sysexrc** file. Clicking **Cancel** leaves options unchanged.

**verbose** Display messages about what is currently taking place.

**verify** Run through a cursory pass of tests to see all subsystems present.

**run_on_err**
Halt subsystem testing when an error occurs.

**sysex_halt_on_err**
Stop **sysex** if an error occurs in any subsystem.

**core** Create core dump in */var/sysex*.

**single_pass**
Run one pass of each selected device test.

For the expert user, more commands are available by clicking the manufacturing cycle to Enabled. This displays the following options:

**fdc_wait** Variable wait time between executions of the diskette test.

**vmem_wait**
Variable delay between successive executions of the virtual memory test.

**debug** Display all messsages to aid analysis of problem systems.

**check_eeprom**
Read NVRAM configuration information and display values.

**intervention**
Turn on or off the confirmer for all destructive tests, or for tests requiring media.

FILES

**/usr/sysex/sysex**
**/var/sysex/core**
**/var/sysex/sysex.info**
**/var/sysex/sysex.error**

NAME

syswait – execute a command, suspending termination until user input

SYNOPSIS

**syswait** *message command*

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**syswait** executes a specified command, suspending termination until the user types any character. *message* is the message prompting the user to type a character to terminate the command. *command* is the command to be executed.

EXAMPLE

The following example invokes a cmdtool and executes 'ls *.c', but waits for the user to type a character before terminating the ls and closing the cmdtool window.

**cmdtool syswait "Press any key to quit..." "ls *.c" &**

## NAME

tabs – set tab stops on a terminal

## SYNOPSIS

**/usr/5bin/tabs** [ *tabspec* ] [ **–T***type* ] [ **+m***n* ]

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**tabs** can be used to specify TAB stops on terminals that support remotely-settable hardware TAB characters. TAB stops are set according to the *tabspec* option, as described below, and previous settings are erased.

Four types of tab specification are accepted for *tabspec*. They are:

*–code* Set the TAB stops according to the canned TAB setting specified by *code*, as given by **fspec**(5).

*–n* Set the TAB stops at intervals of *n* columns, that is, at 1+*n*, 1+2**n*, and so on, as per the *–n* specification as given by **fspec**(5).

*n1,n2,...*
Set the TAB stops at positions *n1*, *n2*, and so on, as per the *n1,n2,...* specification as given by **fspec**(5).

*––file* Read the first line of the file specified by *file*, searching for a format specification as given by **fspec**(5). If this line contains a format specification, set the TAB stops accordingly, otherwise set them to every 8 columns. This type of specification may be used to make sure that a file containing a TAB specification is displayed with correct TAB settings. For example, it can be used with the **pr**(1V) command:

**tabs ––file; pr file**

If no *tabspec* is given, the default value is **–8**, the standard default TAB setting. The lowest column number is 1. Note: for **tabs**, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, (such as the DASI 300, DASI 300s, and DASI 450). TAB and margin setting is performed by echoing to the proper sequences to the standard output.

## OPTIONS

**–T***type* **tabs** usually needs to know the type of terminal in order to set TAB characters, and always needs to know to set margins. *type* is a name listed in **term**(5). If no **–T** flag is supplied, **tabs** uses the value of the environment variable **TERM**. If **TERM** is not defined in the environment (see **environ**(5V)), **tabs** tries a default sequence that will work for many terminals.

**+m***n* The margin argument may be used for some terminals. It moves all TAB stops over *n* columns by making column *n*+1 the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly. **pr**(1V), **tput**(1V), **fspec**(5), **terminfo**(5V), **environ**(5V), **term**(5)

## BUGS

There is no consistency between different terminals regarding ways of clearing tabs and setting the left margin.

**tabs** clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

NAME

tail – display the last part of a file

SYNOPSIS

**tail** +|–*number* [ **lbc** ] [ **fr** ] [ *filename* ]

DESCRIPTION

**tail** copies *filename* to the standard output beginning at a designated place. If no file is named, the standard input is used.

OPTIONS

Options are all jammed together, not specified separately with their own '–' signs.

+*number*

Begin copying at distance *number* from the beginning of the file. *number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines. If *number* is not specified, the value 10 is used.

–*number*

Begin copying at distance *number* from the end of the file. *number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines. If *number* is not specified, the value 10 is used.

**l** *number* is counted in units of lines.

**b** *number* is counted in units of blocks.

**c** *number* is counted in units of characters.

**r** Copy lines from the end of the file in reverse order. The default for **r** is to print the entire file in reverse order.

**f** If the input file is not a pipe, do not terminate after the line of the input file has been copied, but enter an endless loop, sleeping for a second and then attempting to read and copy further records from the input file. This option may be used to monitor the growth of a file that is being written by some other process. For example, the command:

**tail –f fred**

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed. As another example, the command:

**tail –15cf fred**

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time **tail** is initiated and killed.

SEE ALSO

**dd**(1)

BUGS

Data for a tail relative to the end of the file is stored in a buffer, and thus is limited in size.

Various kinds of anomalous behavior may happen with character special files.

NAME

talk – talk to another user

SYNOPSIS

**talk** *person* [ *ttyname* ]

DESCRIPTION

**talk** is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on your own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is one of the following forms :

*host*!*user*
*host*.*user*
*host*:*user*
*user*@*host*

though *user*@*host* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, **talk** sends the message:

**Message from TalkDaemon@***his_machine* **at***time*...
**talk: connection requested by** *your_name*@*your_machine*.
**talk: respond with: talk** *your_name*@*your_machine*

to the user you wish to talk to. At this point, the recipient of the message should reply by typing:

**example% talk** *your_name*@*your_machine*

It does not matter from which machine the recipient replies, as long as their login name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing CTRL–**L** redraws the screen, while your erase, kill, and word kill characters will work in **talk** as normal. To exit, just type your interrupt character; **talk** then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the **mesg** command. At the outset talking is allowed. Certain commands, in particular **nroff**(1) and **pr**(1V) disallow messages in order to prevent messy output.

FILES

| | |
|---|---|
| **/etc/hosts** | to find the recipient's machine |
| **/etc/utmp** | to find the recipient's tty |

SEE ALSO

**mail**(1), **mesg**(1), **nroff**(1), **pr**(1V), **who**(1), **write**(1), **talkd**(8C)

## NAME

tar – create tape archives, and add or extract files

## SYNOPSIS

**tar** [ – ] **crtux** [ **bBefFhilmopvwX014578** ] [ *tarfile* ] [ *blocksize* ] [ *exclude-file* ] [ **–I** *include-file* ] *filename1 filename2* ... **–C** *directory filenameN* ...

## DESCRIPTION

**tar** archives and extracts multiple files onto a single **tar**, file archive, called a *tarfile*. A tarfile is usually a magnetic tape, but it can be any file. **tar**'s actions are controlled by the first argument, the *key*, a string of characters containing exactly one function letter from the set **crtux**, and one or more of the optional function modifiers listed below. Other arguments to **tar** are file or directory names that specify which files to archive or extract. In all cases, the appearance of a directory name refers recursively to the files and subdirectories of that directory.

## FUNCTION LETTERS

**c** Create a new *tarfile* and write the named files onto it.

**r** Write the named files on the end of the *tarfile*. Note: this option *does not work* with quarter-inch archive tapes.

**t** List the table of contents of the *tarfile*.

**u** Add the named files to the *tarfile* if they are not there or if they have been modified since they were last archived. Note: this option *does not work* with quarter-inch archive tapes.

**x** Extract the named files from the *tarfile*. If a named file matches a directory with contents written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no *filename* arguments are given, all files in the archive are extracted. Note: if multiple entries specifying the same file are on the tape, the last one overwrites all earlier versions.

## FUNCTION MODIFIERS

**b** Use the next argument as the blocking factor for tape records. The default blocking factor is 20 blocks. The block size is determined automatically when reading tapes (key letters **x** and **t**). This determination of the blocking factor may be fooled when reading from a pipe or a socket (see the **B** key letter below). The maximum blocking factor is determined only by the amount of memory available to **tar** when it is run. Larger blocking factors result in better throughput, longer blocks on nine-track tapes, and better media utilization.

**B** Force **tar** to perform multiple reads (if necessary) so as to read exactly enough bytes to fill a block. This option exists so that **tar** can work across the Ethernet, since pipes and sockets return partial blocks even when more data is coming.

**e** If any unexpected errors occur **tar** will exit immediately with a positive exit status.

**f** Use the next argument as the name of the *tarfile*. If **f** is omitted, use the device indicated by the **TAPE** environment variable, if set. Otherwise, use **/dev/rmt8** by default. If *tarfile* is given as '–', **tar** writes to the standard output or reads from the standard input, whichever is appropriate. Thus, **tar** can be used as the head or tail of a filter chain. **tar** can also be used to copy hierarchies with the command:

**example% cd fromdir; tar cf – . | (cd todir; tar xfBp –)**

**F** With one **F** argument specified, exclude all directories named SCCS from *tarfile*. With two arguments **FF**, exclude all directories named SCCS, all files with **.o** as as their suffix, and all files named **errs**, **core**, and **a.out**.

**h** Follow symbolic links as if they were normal files or directories. Normally, **tar** does not follow symbolic links.

**i** Ignore directory checksum errors.

**l** Display error messages if all links to archived files cannot be resolved. If **l** is not used, no error messages are printed.

**m** Do not extract modification times of extracted files. The modification time will be the time of extraction.

**o** Suppress information specifying owner and modes of directories which **tar** normally places in the archive. Such information makes former versions of **tar** generate an error message like:

*<filename>/*: **cannot create**

when they encounter it.

**p** Restore the named files to their original modes, ignoring the present **umask**(2). SetUID and sticky information are also extracted if you are the super-user. This option is only useful with the **x** key letter.

**v** Verbose. Normally **tar** does its work silently; this option displays the name of each file **tar** treats, preceded by the function letter. When used with the **t** function, **v** displays the *tarfile* entries in a form similar to '**ls –l**'.

**w** Wait for user confirmation before taking the specified action. If you use **w**, **tar** displays the action to be taken followed by the file name, and then waits for a **y** response to proceed. No action is taken on the named file if you type anything other than a line beginning with **y**.

**X** Use the next argument as a file containing a list of named files (or directories) to be excluded from the *tarfile* when using the key letters **c**, **x**, or **t**. Multiple **X** arguments may be used, with one *exclude file* per argument.

**014578** Select an alternate drive on which the tape is mounted. The numbers **2**, **3**, **6**, and **9** do not specify valid drives. The default is **/dev/rmt8**.

If a file name is preceded by **–I** then the filename is opened. A list filenames, one per line, is treated as if each appeared separately on the command line. Be careful of trailing white space in both include and exclude file lists.

In the case where excluded files (see **X** flag) also exist, excluded files take precedence over all included files. So, if a file is specified in both the include and exclude files (or on the command line), it will be excluded.

If a file name is preceded by **–C** in a **c** (create) or **r** (replace) operation, **tar** will perform a **chdir** (see **csh**(1)) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. See EXAMPLES below.

Note: the **–C** option only applies to *one* following directory name and *one* following file name.

EXAMPLES

To archive files from **/usr/include** and from **/etc**, one might use:

**example% tar c –C /usr include –C /etc .**

If you get a table of contents from the resulting *tarfile*, you will see something like:

**include/**
**include/a.out.h**
*and all the other files in* **/usr/include . . ./chown**
*and all the other files in* **/etc**

Here is a simple example using **tar** to create an archive of your home directory on a tape mounted on drive **/dev/rmt0**:

**example% cd**
**example% tar cvf /dev/rmt0 .**
*messages from* **tar**

The **c** option means create the archive; the **v** option makes **tar** tell you what it is doing as it works; the **f** option means that you are specifically naming the file onto which the archive should be placed (**/dev/rmt0** in this example).

Now you can read the table of contents from the archive like this:

```
example% tar tvf /dev/rmt0          display table of contents of the archive
(access user-id/group-id   size   mod. date          filename)
rw-r--r-- 1677/40          2123   Nov 7 18:15:1985   ./archive/test.c
...
example%
```

You can extract files from the archive like this:

```
example% tar xvf /dev/rmt0          extract files from the archive
messages from tar
example%
```

If there are multiple archive files on a tape, each is separated from the following one by an EOF marker. **tar** does not read the EOF mark on the tape after it finishes reading an archive file because **tar** looks for a special header to decide when it has reached the end of the archive. Now if you try to use **tar** to read the next archive file from the tape, **tar** does not know enough to skip over the EOF mark and tries to read the EOF mark as an archive instead. The result of this is an error message from **tar** to the effect:

```
tar: blocksize=0
```

This means that to read another archive from the tape, you must skip over the EOF marker before starting another **tar** command. You can accomplish this using the *mt* command, as shown in the example below. Assume that you are reading from **/dev/nrmt0**.

```
example% tar xvfp /dev/nrmt0        read first archive from tape
messages from tar
example% mt fsf 1                   skip over the end-of-file marker
example% tar xvfp /dev/nrmt0        read second archive from tape
messages from tar
example%
```

Finally, here is an example using **tar** to transfer files across the Ethernet. First, here is how to archive files from the local machine (**example**) to a tape on a remote system (**host**):

```
example% tar cvfb - 20 filenames | rsh host dd of=/dev/rmt0 obs=20b
messages from tar
example%
```

In the example above, we are *creating* a *tarfile* with the **c** key letter, asking for *verbose* output from **tar** with the **v** option, specifying the name of the output *tarfile* using the **f** option (the standard output is where the *tarfile* appears, as indicated by the '–' sign), and specifying the blocksize (20) with the **b** option. If you want to change the blocksize, you must change the blocksize arguments both on the **tar** command *and* on the **dd** command.

Now, here is how to use **tar** to get files from a tape on the remote system back to the local system:

```
example% rsh -n host dd if=/dev/rmt0 bs=20b | tar xvBfb - 20 filenames
messages from tar
example%
```

In the example above, we are *extracting* from the *tarfile* with the **x** key letter, asking for *verbose output from* **tar** with the **v** option, telling **tar** it is reading from a pipe with the **B** option, specifying the name of the input *tarfile* using the **f** option (the standard input is where the *tarfile* appears, as indicated by the '–' sign), and specifying the blocksize (20) with the **b** option.

FILES

| | |
|---|---|
| **/dev/rmt?** | half-inch magnetic tape interface |
| **/dev/rar?** | quarter-inch magnetic tape interface |
| **/dev/rst?** | SCSI tape interface |
| **/tmp/tar*** | |

ENVIRONMENT

**TAPE** If specified, in the environment, the value of **TAPE** indicates the default tape device.

SEE ALSO

**cpio**(1), **csh**(1), **umask**(2), **tar**(5), **dump**(8), **restore**(8)

BUGS

Neither the **r** option nor the **u** option can be used with quarter-inch archive tapes, since these tape drives cannot backspace.

There is no way to ask for the *n*th occurrence of a file.

Tape errors are handled ungracefully.

The **u** option can be slow.

There is no way selectively to follow symbolic links.

When extracting tapes created with the **r** or **u** options, directory modification times may not be set correctly.

Files with names longer that 100 characters cannot be processed.

Filename substitution wildcards do not work for extracting files from the archive. To get around this, use a command of the form:

**tar xvf... /dev/rst0 'tar tf... /dev/rst0 | grep '*pattern*'**

## NAME

tbl – format tables for nroff or troff

## SYNOPSIS

**tbl** [ **–ms** ] [ **–mm** ] [ *filename* ] ...

## AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**tbl** is a preprocessor for formatting tables for **nroff**(1) or **troff**(1). The input *filename*s are copied to the standard output, except that lines between **.TS** and **.TE** command lines are assumed to describe tables and are reformatted. Details are given in *Formatting Documents*.

If no arguments are given, **tbl** reads the standard input, so **tbl** may be used as a filter. When **tbl** is used with **eqn**(1) or **neqn** the **tbl** command should be first, to minimize the volume of data passed through pipes.

## OPTIONS

**–ms** Copy the **–ms** macro package to the front of the output file.

**–mm** Copy the **–mm** macro package to the front of the output file.

## EXAMPLE

As an example, letting represent a TAB (which should be typed as a genuine TAB) the input

```
""
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

| Household Population | | |
|---|---|---|
| Town | Households | |
| | Number | Size |
| Bedminster | 789 | 3.26 |
| Bernards Twp. | 3087 | 3.74 |
| Bernardsville | 2018 | 3.30 |
| Bound Brook | 3425 | 3.04 |
| Branchburg | 1644 | 3.49 |
| Bridgewater | 7897 | 3.81 |
| Far Hills | 240 | 3.19 |

**SEE ALSO**

**eqn**(1), **nroff**(1), **troff**(1)

*Formatting Documents*

**NAME**

tcopy – copy a magnetic tape

**SYNOPSIS**

**tcopy** *source* [ *destination* ]

**DESCRIPTION**

**tcopy** copies the magnetic tape mounted on the tape drive specified by the *source* argument. The only assumption made about the contents of a tape is that there are two tape marks at the end.

When only a source drive is specified, **tcopy** scans the tape, and displays information about the sizes of records and tape files. If a destination is specified, **tcopy** makes a copies the source tape onto the *destination* tape, with blocking preserved. As it copies, **tcopy** produces the same output as it does when only scanning a tape.

**SEE ALSO**

**mt**(1)

NAME

tcov – construct test coverage analysis and statement-by-statement profile

SYNOPSIS

**tcov** [ **–a** ] [ *–n* ] *srcfile*...

AVAILABILITY

Sun-2, Sun-3, and Sun-4 systems only.

DESCRIPTION

**tcov** produces a test coverage analysis and statement-by-statement profile of a C or FORTRAN program. When a program in a file named *file*.**c** or *file*.**f** is compiled with the **–a** option, a corresponding *file*.**d** file is created. Each time the program is executed, test coverage information is accumulated in *file*.**d.**

**tcov** takes source files as arguments. It reads the corresponding *file*.**d** file and produces an annotated listing of the program with coverage data in *file*.**tcov.** Each straight-line segment of code (or each line if the **–a** option to **tcov** is specified) is prefixed with the number of times it has been executed; lines which have not been executed are prefixed with #####.

Note: the profile produced includes only the number of times each statement was executed, not execution times; to obtain times for routines use **gprof**(1) or **prof**(1).

OPTIONS

**–a** Display an execution count for each statement; if **–a** is not specified, an execution count is displayed only for the first statement of each straight-line segment of code.

*–n* Display table of the line numbers of the *n* most frequently executed statements and their execution counts.

EXAMPLES

The command:

**example% cc -a -o prog prog.c**

compiles with the **-a** option — produces **prog.d**

The command: **example% prog**

executes the program '–' accumulates data in **prog.d**

The command:

**example% tcov prog.c** produces an annotated listing in file **prog.tcov**

FILES

| | |
|---|---|
| **file.c** | input C program file |
| **file.f** | input FORTRAN program file |
| **file.d** | input test coverage data file |
| **file.tcov** | output test coverage analysis listing file |
| **/usr/lib/bb_link.o** | entry and exit routines for test coverage analysis |

SEE ALSO

**cc**(1V), **gprof**(1), **prof**(1), **exit**(2)

DIAGNOSTICS

**premature end of file**

Issued for routines containing no statements.

BUGS

The analyzed program must call **exit**(2) or return normally for the coverage information to be saved in the *.d* file.

NAME

tee – replicate the standard output

SYNOPSIS

**tee** [ **–ai** ] [ *filename* ] . . .

DESCRIPTION

**tee** transcribes the standard input to the standard output and makes copies in the *filename*s.

OPTIONS

**–a** Append the output to the *filename*s rather than overwriting them.

**–i** Ignore interrupts.

## NAME

telnet – user interface to a remote system using the TELNET protocol

## SYNOPSIS

**telnet** [ *host* [ *port* ] ]

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**telnet** communicates with another host using the TELNET protocol. If **telnet** is invoked without arguments, it enters command mode, indicated by its prompt (**telnet>**). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, **telnet** enters input mode. In this mode, text typed is sent to the remote host. The input mode entered will be either "character at a time" or "line by line" depending on what the remote system supports.

In "character at a time" mode, most text typed is immediately sent to the remote host for processing.

In "line by line" mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The "local echo character" (initially "^E') may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's **quit, intr,** and **flush** characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see **toggle autoflush** and **toggle autosynch** below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of **quit** and **intr**).

While connected to a remote host, **telnet** command mode may be entered by typing the **telnet** "escape character" (initially '^]', (control-right-bracket)). When in command mode, the normal terminal editing conventions are available.

## USAGE

### Telnet Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode, set, toggle,** and **display** commands).

**open** *host* [ *port* ]
: Open a connection to the named host. If no port number is specified, **telnet** will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see **hosts**(5)) or an Internet address specified in the "dot notation" (see **inet**(3N)).

**close**
: Close a TELNET session and return to command mode.

**quit**
: Close any open TELNET session and exit **telnet**. An EOF (in command mode) will also close a session and exit.

**z**
: Suspend **telnet**. This command only works when the user is using a shell that supports job control, such as **csh**(1).

**mode** *type*
: *type* is either **line** (for "line by line" mode) or *character* (for "character at a time" mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

**status**
: Show the current status of **telnet**. This includes the peer one is connected to, as well as the current mode.

**display** [ *argument...* ]
Display all, or some, of the **set** and **toggle** values (see below).

**?** [ *command* ]
Get help. With no arguments, **telnet** print a help summary. If a command is specified, **telnet** will print the help information for just that command.

**send** *arguments*
Send one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

*escape* Send the current **telnet** escape character (initially "^]").

*synch* Send the TELNET **SYNCH** sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it does not work, a lower case "r" may be echoed on the terminal).

**brk** Send the TELNET **BRK** (Break) sequence, which may have significance to the remote system.

**ip** Send the TELNET **IP** (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

*ao* Sends the TELNET **AO** (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

*ayt* Sends the TELNET **AYT** (Are You There) sequence, to which the remote system may or may not choose to respond.

**ec** Sends the TELNET **EC** (Erase Character) sequence, which should cause the remote system to erase the last character entered.

*el* Sends the TELNET **EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

*ga* Sends the TELNET **GA** (Go Ahead) sequence, which likely has no significance to the remote system.

*nop* Sends the TELNET **NOP** (No Operation) sequence.

*?* Prints out help information for the **send** command.

**set** *argument value*
Set any one of a number of **telnet** variables to a specific value. The special value "off" turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

**echo** This is the value (initially "^E") which, when in "line by line" mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

*escape* This is the **telnet** escape character (initially "^[") which causes entry into **telnet** command mode (when connected to a remote system).

*interrupt*
If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *interrupt* character is typed, a TELNET **IP** sequence (see **send ip** above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

*quit* If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed, a TELNET **BRK** sequence (see **send brk** above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

*flushoutput*
If **telnet** is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed, a TELNET AO sequence (see **send** *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

*erase* If **telnet** is in *localchars* mode (see **toggle** *localchars* below), **and** if **telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET EC sequence (see **send ec** above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

**kill** If **telnet** is in *localchars* mode (see **toggle** *localchars* below), **and** if **telnet** is operating in "character at a time" mode, then when this character is typed, a TELNET EL sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

*eof* If **telnet** is operating in "line by line" mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

**toggle** *arguments...*
Toggle (between TRUE and FALSE) various flags that control how **telnet** responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

*localchars*
If this is TRUE, then the **flush, interrupt, quit, erase,** and **kill** characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in "line by line" mode, and FALSE in "character at a time" mode.

*autoflush*
If *autoflush* and *localchars* are both TRUE, then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into TELNET sequences; see **set** above for details), **telnet** refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET *Timing Mark* option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh", otherwise FALSE (see **stty(1V)**).

*autosynch*
If *autosynch* and *localchars* are both TRUE, then when either the *intr* or *quit* characters is typed (see **set** above for descriptions of the *intr* and *quit* characters), the resulting TELNET sequence sent is followed by the TELNET SYNCH sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

*crmod* Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

*debug* Toggle socket level debugging (useful only to the super-user). The initial value for this toggle is FALSE.

*options* Toggle the display of some internal **telnet** protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

*netdata* Toggle the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

*?* Display the legal **toggle** commands.

SEE ALSO

**csh**(1), **rlogin**(1C), **stty**(1V) **inet**(3N), **hosts**(5)

BUGS

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in ''line by line'' mode.

There is enough settable state to justify a **.telnetrc** file.

No capability for a **.telnetrc** file is provided.

In ''line by line'' mode, the terminal's EOF character is only recognized (and sent to the remote system) when it is the first character on a line.

## NAME

test – return true or false according to a conditional expression

## SYNOPSIS

**test** *expression*

[ *expression* ]

## DESCRIPTION

**test** evaluates the expression *expression* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned. **test** returns a non-zero exit if there are no arguments.

## SYSTEM V DESCRIPTION

The actions of the System V version of **test** are the same, except for the following primitives:

**–f** *filename* True if *filename* exists and is a regular file.

**–l** *string* Not supported.

## USAGE

### Primitives

The following primitives are used to construct *expression*.

**–b** *filename*
True if *filename* exists and is a block special device.

**–c** *filename*
True if *filename* exists and is a character special device.

**–d** *filename*
True if *filename* exists and is a directory.

**–f** *filename*
True if *filename* exists and is not a directory.

**–g** *filename*
True if *filename* exists and its set-group-ID bit is set.

**–h** *filename*
True if *filename* exists and is a symbolic link.

**–k** *filename*
True if *filename* exists and its sticky bit is set.

**–l** *string* the length of the string.

**–n** *s1* True if the length of the string *s1* is non-zero.

**–p** *filename*
True if *filename* exists and is a named pipe (FIFO).

**–r** *filename*
True if *filename* exists and is readable.

**–s** *filename*
True if *filename* exists and has a size greater than zero.

**–t** [ *fildes* ]
True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

**–u** *filename*
True if *filename* exists and its set-user-ID bit is set.

**–w** *filename*
True if *filename* exists and is writable.

**-x** *filename*
True if *filename* exists and is executable.

**-z** *s1* True if the length of string *s1* is zero.

*s1* **=** *s2* True if the strings *s1* and *s2* are equal.

*s1* **!=** *s2* True if the strings *s1* and *s2* are not equal.

*s1* True if *s1* is not the null string.

*n1* **-eq** *n2* True if the integers *n1* and *n2* are algebraically equal.

*n1* **-ne** *n2* True if *n1* is not equal to *n2*.

*n1* **-gt** *n2* True if *n1* is greater than *n2*.

*n1* **-ge** *n2* True if *n1* is greater than or equal to *n2*.

*n1* **-lt** *n2* True if *n1* is less than *n2*.

*n1* **-le** *n2* True if *n1* is less than or equal to *n2*.

**Operators**

The above primaries may be combined with the following operators:

**!** Unary negation operator.

**-a** Binary *and* operator.

**-o** Binary *or* operator.

(*expression*)
Parentheses for grouping.

**-a** has higher precedence than **-o.** Notice that all the operators and flags are separate arguments to **test**. Notice also that parentheses are meaningful to the Shell and must be escaped.

## SEE ALSO

**find**(1), **sh**(1)

## WARNING

In the second form of the command (that is, the one that uses [ ], rather than the word **test**), the square brackets must be delimited by blanks.

Some UNIX systems do not recognize the second form of the command.

## NOTE

The **test** command is built into the Bourne shell, which chooses the 4.2 BSD or the System V version of **test**, depending on whether **/usr/5bin** appears before **/usr/bin** in the shell's **PATH** variable. This is consistent with the behavior of other commands present in both **/usr/bin** and **/usr/5bin**.

The fact that **test** is built into the shell also means that a program named **test** cannot be run without specifying a pathname; if the program is in the current directory, **./test** will suffice.

NAME

textedit – SunView window- and mouse-based text editor

SYNOPSIS

**textedit** [ *generic-tool-arguments* ] [ **–Ea on | off** ] [ **–adjust_is_pending_delete** ] [ **–Ei on | off** ]
[ **–auto_indent** ] [ **–Eo on | off** ] [ **–okay_to_overwrite** ] [ **–Er on | off** ] [ **–read_only** ]
[ **–Ec** *N* ] [ **–checkpoint** *count* ] [ **–EL** *lines* ] [ **–lower_context** *lines* ] [ **–Em** *pixels* ]
[ **–margin** *pixels* ] [ **–En** *N* ] [ **–number_of_lines** *lines* ] [ **–Es** *N* ] [ **–scratch_window** *lines* ]
[ **–ES** *N* ] [ **–multi_click_space** *radius* ] [ **–Et** *N* ] [ **–tab_width** *tabstop* ] [ **–ET** *N* ]
[ **–multi_click_timeout** *intrvl* ] [ **–Eu** *N* ] [ **–history_limit** *max* ] [ **–EU** *N* ]
[ **–upper_context** *lines* ] *filename*

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**textedit** is a mouse-oriented text editor that runs within the SunView environment. It creates a window containing two text subwindows. The top subwindow (referred to as the "scratch" window) can be used to store small pieces of text. The bottom subwindow (referred to as the "edit" window) displays the contents of *filename*, if given.

The name of the file currently being edited is displayed in the left-hand portion of the frame header. The name of the current working directory is displayed in the right-hand portion.

OPTIONS

*generic-tool-arguments*
**textedit** accepts the SunView generic tool arguments listed in **sunview**(1).

**–Ea on | off**
**–adjust_is_pending_delete**
Choose whether or not an adjustment to a selection makes the selection "pending-delete." The default is off. This option corresponds to, and overrides, the **adjust_is_pending_delete** Text defaults entry.

**–Ei on | off**
**–auto_indent** Choose whether or not to automatically indent newly-opened lines. The default is off. Corresponds to the **auto_indent** Text default.

**–Eo on | off**
**–okay_to_overwrite**
Set behavior to the **Store as New File** menu item. If **on** a **Store as New File** to the current file is treated as a **Save Current File**. If **off** (the standard default), **Store as New File** operations using the current filename result in an error message. Corresponds to **Store_self_is_save**.

**–Er on | off**
**–read_only** Turn read-only mode on or off. When on, text cannot be modified.

**–Ec** *N*
**–checkpoint** *count*
Checkpoint after every *count* editing operations. If *count* is 0 (the standard default), no checkpointing takes place. Each character typed, each **Paste**, and each **Cut** counts as an editing operation. Corresponds to **checkpoint_frequency**.

**–EL** *lines*
**–lower_context** *lines*
Specify the minimum number of lines to keep between the caret and the bottom of the text subwindow. The default is 2. Corresponds to **lower_context**.

**–Em** *pixels*
**–margin** *pixels*
Set the scrollbar margin width in pixels. The default is 4. Corresponds to **left_margin**.

**–En** *N*
**–number_of_lines** *lines*
Set the number of lines in the bottom subwindow. The default is 45.

**–Es** *N*
**–scratch_window** *lines*
Set the number of lines in the scratch window. A zero value means that there is no scratch window. The standard default is 1. Corresponds to **scratch_window**.

**–ES** *N*
**–multi_click_space** *radius*
Set the radius, in pixels, within which clicks must occur to be treated as a multi-click selection. The default is 3 pixels. Corresponds to **multi_click_space**.

**–Et** *N*
**–tab_width** *tabstop*
Set the number of SPACE characters displayed per TAB stop. The default is 8. This option has no effect on the characters in the file. Corresponds to **tab_width**.

**–ET** *N*
**–multi_click_timeout** *intrvl*
Set the interval, in milliseconds, within which any two clicks must occur to be treated as a multi-click selection. The default is 390 milliseconds. Corresponds to **multi_click_timeout**.

**–Eu** *N*
**–history_limit** *max*
Set the maximum number of editing operations that can be undone or replayed. The default is 50. Corresponds to **history_limit**.

**–EU N**
**–upper_context** *lines*
Set the minimum number of lines to keep between the caret and the top of the text subwindow. The default is 2. Corresponds to **upper_context**.

## USAGE

For a description of how to use the facilities of the text subwindows, see the *SunView 1 Beginner's Guide*.

### Signal Processing

If **textedit** hangs, for whatever reason, you can send a SIGHUP signal to its process ID, which forces it to write any changes (if possible):

**kill –HUP** *pid*

The edits are written to the file **textedit.***pid* in its working directory. If that fails, **textedit** successively tries to write to a file by that name in **/var/tmp**, and then **/tmp**. In addition, whenever **textedit** catches a fatal signal, such as SIGILL, it tries to write out the edits before aborting.

### Defaults Options

There are several dozen user-specified defaults that affect the behavior of the text-based facilities. See **defaultsedit**(1) for a complete description. Important defaults entries in the **Text** category are:

**Edit_back_char** Set the character for erasing to the left of the caret. The standard default is DELETE. Note: the tty erase character-setting has no effect on **textedit**. Text-based tools refer only to the defaults database key settings.

**Edit_back_word** Set the character for erasing the word to the left of the caret. The standard default is CTRL-W.

**Edit_back_line** Set the character for erasing all characters to the left of the caret. The standard default is CTRL-U.

**Checkpoint_frequency**
If set to **0** (the standard default) no checkpointing is done. For any value greater than zero, a checkpoint is made each time the indicated number of editing operations has been performed since the last checkpoint. Each character typed, each **Paste**, and each **Cut** counts as an editing operation. The checkpoint file has a name of the form: *filename* **%%**, where *filename* is the name of the file being edited.

**Making a selection**

In **textedit**, the mouse is used to specify a selection, which is a character span to operate on. The mouse is also used to position the insertion point and to invoke a menu of commands.

The assignment of commands to the mouse buttons is:

| Mouse button | Description |
|---|---|
| LEFT | Starts a new selection and moves the insertion point to the end of the selection nearest the mouse cursor. |
| MIDDLE | Extends a selection, and moves the insertion point. |
| RIGHT | Displays a menu of operations, explained below. |

There are two types of selections: a primary selection is indicated by video-inversion of the span of characters, and tends to persist. A secondary selection is indicated by underlining the span of characters and only exists while one of the four function keys corresponding to the commands **Cut**, **Find**, **Paste**, or **Copy**, is depressed.

In addition, a selection can be "pending-delete," as indicated by overlaying the span of characters with a light gray pattern. A selection is made pending-delete by holding the CTRL key while clicking the LEFT or MIDDLE mouse buttons. If a primary selection is pending-delete, it is only deleted when characters are inserted, either by type-in or by **Paste** or **Copy**. If a secondary selection is pending-delete, it is deleted when the function key is released, except in the case of the **Find**, which deselects the secondary selection.

You can make adjusted selections switch to pending-delete using the **adjust_is_pending_delete** defaults entry, or the **-Ea** option. In this case, **CTRL-Middle** makes the selection *not* pending-delete.

Commands that operate on the primary selection do so even if the primary selection is not in the window that issued the command.

**Inserting Text and Command Characters**

For the most part, typing any of the standard keys either inserts the corresponding character at the insertion point, or erases characters. However, certain key combinations are treated as commands. Some of the most useful are:

| Command | Character | Description |
|---|---|---|
| **Cut-Primary** | Meta-X | Erases, and moves to the Clipboard, the primary selection. |
| **Find-Primary** | Meta-F | Searches the text for the pattern specified by the primary selection or by the Clipboard, if there is no primary selection. |
| **Copy-to-Clipboard** | Meta-C | Copies the primary selection to the Clipboard. |
| **Paste-Clipboard** | Meta-V | Inserts the Clipboard contents at the insertion point. |
| **Copy-then-Paste** | Meta-P | Copies the primary selection to the insertion point (through the Clipboard). |
| **Go-to-EOF** | CTRL-RETURN | Moves the insertion point to the end of the text, positioning the text so that the insertion point is visible. |

**Function Keys**

The commands indicated by use of the function keys are:

| Command | Sun-2\|3 Key | Description |
|---|---|---|
| **Stop** | L1 | Aborts the current command. |
| **Again** | L2 | Repeats the previous editing sequence since a primary selection was made. |
| **Undo** | L4 | Undoes a prior editing sequence. |
| **Front** | L5 | Makes the window completely visible (or hides it, if it is already exposed). |
| **Copy** | L6 | Copies the primary selection, either to the Clipboard or at the closest end of the secondary selection. |
| **Open** | L7 | Makes the window iconic (or normal, if it is already iconic). |
| **Paste** | L8 | Copies either the secondary selection or the Clipboard at the insertion point. |
| **Find** | L9 | Searches for the pattern specified by, in order, the secondary selection, the primary selection, or the Clipboard. |
| **Cut** | L10 | Erases, and moves to the Clipboard, either the primary or the secondary selection. |
| CAPSLOCK | F1 | Forces all subsequently typed alphabetic characters to be upper-case. This key is a toggle; striking it a second time undoes the effect of the first strike. |

**Find** usually searches the text forwards, towards the end. Holding down the SHIFT key while invoking **Find** searches backward through the text, towards the beginning. If the pattern is not found before the search encounters either extreme, it "wraps around" and continues from the other extreme. **Find** starts the search at the appropriate end of the primary selection, if the primary selection is in the subwindow that the search is made in; otherwise it starts at the insertion point, unless the subwindow cannot be edited, in which case it starts at the beginning of the text.

CTRL-**Find** invokes the **Find and Replace** pop-up frame.

The default assignment of function keys can be modified using **defaultsedit**(1).

**Menu Items**

**File** A pull-right menu item for file operations.

**Edit** A pull-right menu item equivalent of the editing function keys. The **Edit** submenu provides **Again, Undo, Copy, Paste,** and **Cut** (same as function keys L2, L4, L6, L8, and L10).

**Display** A pull-right menu item for controlling the way text is displayed and line display format.

**Find** A pull-right menu item for find and delimiter matching operations.

**Extras** A user definable pull-right menu item. The **Extras** standard submenu is controlled by **/usr/lib/text_extras_menu**. This file has the same syntax as **.rootmenu** file. See **sunview**(1).

Only those items that are active appear as normal text in the menu; inactive items (which are inappropriate at the time) are "grayed out".

**User Defined Commands**

The file **/usr/lib/text_extras_menu** specifies filter programs that are included in the text subwindow **Extras** pull-right menu item. The file **/.textswrc** specifies filter programs that are assigned to (available) function keys. These filters are applied to the contents of the primary selection. Their output is entered at the caret.

The file **/usr/lib/textswrc** is a sample containing a set of useful filters. It is not read automatically.

FILES

| | |
|---|---|
| **/.textswrc** | specifies bindings of filters to function keys |
| **/usr/lib/text_extras_menu** | specifies bindings of filters for the extras menu pull-right items |
| **/usr/bin** | contains useful filters, including **shift_lines** and **capitalize**. |
| *filename%* | prior version of *filename* is available here after a **Save Current File** menu operation |
| **textedit.***pid* | edited version of *filename*; generated in response to fatal internal errors |
| **/tmp/Text*** | editing session logs |

SEE ALSO

**defaultsedit**(1), **kill**(1), **sunview**(1), **textswrc**(5)

*SunView 1 Beginner's Guide*

DIAGNOSTICS

**Cannot open file '***filename***', aborting!**

*filename* does not exist or cannot be read.

**textedit** produces the following exit status codes:

| | |
|---|---|
| 0 | normal termination |
| 1 | standard SunView help message was printed |
| 2 | help message was requested and printed |
| 3 | abnormal termination in response to a signal, usually due to an internal error |
| 4 | abnormal termination during initialization, usually due to a missing file or running out of swap space |

BUGS

Multi-click to change the current selection does not work for **Adjust Selection**.

Handling of long lines is incorrect in certain scrolling situations.

There is no way to replay any editing sequence except the most recent.

'**textedit** *newfile*' fails if *newfile* does not exist.

NAME

textedit_filters, align_equals, capitalize, insert_brackets, shift_lines – filters provided with textedit(1)

SYNOPSIS

**align_equals**

**capitalize**

**insert_brackets** *l r*

**shift_lines** *n*

DESCRIPTION

Each of these filters can be mapped to function keys in your **.textswrc** file, and applied to the current selection. See *SunView 1 Beginner's Guide* for details. When one is as a command (perhaps in a pipeline), it is applied to the standard input.

**align_equals** lines up the '=' (equal signs) in C assignment statements. Some programmers feel that this makes for improved readability. It aligns all equal signs with the rightmost equal sign in the selection (or the standard input), by padding with spaces between the sign and the previous nonwhite character; it replaces the selection with the aligned text (or writes this text to the standard output).

For instance:

```
big_long_expression = z;
abc = x;
medium_expression = y;
z += 1;
```

becomes:

```
big_long_expression = z;
abc                 = x;
medium_expression   = y;
z                  += 1;
```

**capitalize** changes the capitalization of the selection (or the standard input) and replaces it (or writes to the standard output). If the text is all capitals, it is converted to all lowercase.

If the text is all lowercase or of mixed cases and contains no white space (such as a NEWLINE, SPACE, or TAB), it is converted to all capitals. If there is white space, then the case of the first character in each word is inverted.

**insert_brackets** surrounds the selection (or the standard input) with brackets. *l* and *r* are left- and right-bracket characters, respectively.

**shift_lines** adjusts indention of the the selection (or the standard input) by *n* spaces, and replaces the selection with the adjusted text (or writes to the standard output). **shift_lines** adjusts to the left when *n* is negative.

FILES

**/tmp/Cap.***pid*
**/tmp/Ins.***pid*

SEE ALSO

**textedit**(1)

*SunView 1 Beginner's Guide*

## NAME

tftp – trivial file transfer program

## SYNOPSIS

**tftp** [ *host* ]

## AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**tftp** is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case **tftp** uses *host* as the default host for future transfers (see the **connect** command below).

## USAGE

### Commands

Once **tftp** is running, it issues the prompt: **tftp>** and recognizes the following commands:

**connect** *host-name* [ *port* ]
Set the *host* (and optionally *port*) for transfers. Note: the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the **connect** command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the **connect** command; the remote host can be specified as part of the **get** or **put** commands.

**mode** *transfer-mode*
Set the mode for transfers; *transfer-mode* may be one of **ascii** or **binary**. The default is **ascii**.

**put** *filename*
**put** *localfile remotefile*
**put** *filename1 filename2 ... filenameN remote-directory*
Transfer a file, or a set of files, to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form

*host*:*filename*

to specify both a host and filename at the same time. If the latter form is used, the specified host becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be running the UNIX system.

**get** *filename*
**get** *remotename*
*localname*
**get** *filename1 filename2 ... filenameN*
Get a file or set of files from the specified remote *sources*. *source* can be in one of two forms: a filename on the remote host if the host has already been specified, or a string of the form

*host*:*filename*

to specify both a host and filename at the same time. If the latter form is used, the last host specified becomes the default for future transfers.

**quit** Exit **tftp**. An EOF also exits.

**verbose** Toggle verbose mode.

**trace** Toggle packet tracing.

**status** Show current status.

**rexmt** *retransmission-timeout*
Set the per-packet retransmission timeout, in seconds.

**timeout** *total-transmission-timeout*
Set the total transmission timeout, in seconds.

**ascii** Shorthand for **mode ascii.**

**binary** Shorthand for **mode binary.**

**?** [ *command-name* ... ]
Print help information.

**WARNING**

The default *transfer-mode* is **ascii.** This differs from pre-4.0 Sun (and pre-4.3 BSD) releases, so explicit action must now be taken when transferring non-ASCII files such as executable commands.

**BUGS**

Because there is no user-login or validation within the TFTP protocol, many remote sites restrict file access in various ways. Approved methods for file access are specific to each site, and therefore cannot be documented here.

NAME

time – time a command

SYNOPSIS

**time** [ *command* ]

DESCRIPTION

There are three distinct versions of **time**: it is built in to the C shell, and is an executable program available in **/usr/bin/time** and **/usr/5bin/time** when using the Bourne shell. In each case, times are displayed on the diagnostic output stream.

In the case of the C shell, a **time** command with no *command* argument simply displays a summary of time used by this shell and its children. When arguments are given the specified simple *command* is timed and the C shell displays a time summary as described in **csh**(1).

The **time** commands in **/usr/bin/time** and **/usr/5bin/time** time the given *command*, which must be specified, that is, *command* is not optional as it is in the C shell's timing facility. When the command is complete, **time** displays the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds. The only difference between the versions in **/usr/bin/time** and **/usr/5bin/time** is between their output formats; **/usr/bin/time** prints all three times on the same line, while **/usr/5bin/time** prints them on separate lines.

EXAMPLES

The three examples here show the differences between the **csh** version of **time** and the versions in **/usr/bin/time** and **/usr/bin/time**. The example assumes that **csh** is the shell in use.

```
example% time wc /usr/share/man/man1/csh.1
1876  11223  65895 /usr/share/man/man1/csh.1
2.7u 0.9s 0:03 91% 3+5k 19+2io 1pf+0w
example%/usr /bin/time wc /usr/share/man/man1/csh.1
1876  11223  65895 /usr/share/man/man1/csh.1
4.3 real       2.7 user       1.0 sys
example% /usr/5bin/time wc /usr/share/man/man1/csh.1
1876  11223  65895 /usr/share/man/man1/csh.1
real      4.3
user      2.7
sys       1.0
example%
```

SEE ALSO

**csh**(1)

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 50th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

BUGS

When the command being timed is interrupted, the timing values displayed may not always be accurate.

## NAME

tip, cu – terminal emulator, telephone connection to a remote system

## SYNOPSIS

**tip** [ **–v** ] [ *–speed-entry* ] *hostname* | *phone-number*

**cu** *phone-number* [ **–t** ] [ **–s** *speed* ] [ **–a** *acu* ] [ **–l** *line* ] [ **–#** ]

## DESCRIPTION

**tip** establishes a full-duplex terminal connection to a remote host. Once the connection is established, a remote session using **tip** behaves like an interactive session on a local terminal.

The preferred interface for remote connections is **tip**; **cu** is included for those who are familiar with the "call UNIX" command of the version 7 UNIX system. This manual page describes only **tip**.

The *remote* file (described in the **remote**(5) manual page) contains entries describing remote systems and line speeds used by **tip**.

Each host has a default baud rate for the connection, or you can specify a speed with the *–speed-entry* command line argument.

When *phone-number* is specified, **tip** looks for an entry in the *remote* file of the form:

**tip** *–speed-entry*

When it finds such an entry, it sets the connection speed accordingly. If it finds no such entry, **tip** interprets *–speed-entry* as if it were a system name, resulting in an error message.

If you omit *–speed-entry*, **tip** uses the **tip0** entry to set a speed for the connection.

When establishing the connection **tip** sends a connection message to the remote system. The default value for this message can be found in the *remote* file.

When **tip** attempts to connect to a remote system, it opens the associated device with an exclusive-open **ioctl**(2) call. Thus only one user at a time may access a device. This is to prevent multiple processes from sampling the terminal line. In addition, **tip** honors the locking protocol used by **uucp**(1C).

When **tip** starts up it reads commands from the file **.tiprc** in your home directory.

## OPTIONS

**–v** Display commands from the **.tiprc** file as they are executed.

## USAGE

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well).

At any time that **tip** prompts for an argument (for example, during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, aborts the dialogue and returns you to the remote machine.

### Commands

A tilde ( ~ ) appearing as the first character of a line is an escape signal which directs **tip** to perform some special action. **tip** recognizes the following escape sequences:

**~^D**

**~.** Drop the connection and exit (you may still be logged in on the remote machine).

**~c** [*name*]
Change directory to *name* (no argument implies change to your home directory).

**~!** Escape to a shell (exiting the shell returns you to **tip**).

**~>** Copy file from local to remote.

**~<** Copy file from remote to local.

~**p** *from* [ *to* ]
Send a file to a remote host running the UNIX system. When you use the put command, the remote system runs the command string

**cat >** *to*

while **tip** sends it the *from* file. If the *to* file is not specified, the *from* file name is used. This command is actually a UNIX-system-specific version of the '~>' command.

~**t** *from* [ *to* ]
Take a file from a remote host running the UNIX system. As in the put command the *to* file defaults to the *from* file name if it is not specified. The remote host executes the command string

**cat** *from* **; echo ^A**

to send the file to **tip**.

~| Pipe the output from a remote command to a local process. The command string sent to the local system is processed by the shell.

~**C** Connect a program to the remote machine. The command string sent to the program is processed by the shell. The program inherits file descriptors 0 as remote line input, 1 as remote line output, and 2 as tty standard error.

~**$** Pipe the output from a local process to the remote host. The command string sent to the local system is processed by the shell.

~**#** Send a BREAK to the remote system.

~**s** Set a variable (see the discussion below).

~^**Z** Stop **tip** (only available when run under a shell that supports job control, such as the C shell).

~^**Y** Stop only the "local side" of **tip** (only available when run under a shell that supports job control, such as the C shell); the "remote side" of **tip**, the side that displays output from the remote host, is left running.

~**?** Get a summary of the tilde escapes.

Copying files requires some cooperation on the part of the remote host. When a ~> or ~< escape is used to send a file, **tip** prompts for a file name (to be transmitted or received) and a command to be sent to the remote system, in case the file is being transferred from the remote system. The default end of transmission string for transferring a file from the local system to the remote is specified as the **oe** capability in the **remote**(5) file, but may be changed by the set command. While **tip** is transferring a file the number of lines transferred will be continuously displayed on the screen. A file transfer may be aborted with an interrupt.

AUTO-CALL UNITS

**tip** may be used to dial up remote systems using a number of auto-call unit's (ACU's). When the remote system description contains the **du** capability, **tip** uses the call-unit (**cu**), ACU type (**at**), and phone numbers (**pn**) supplied. Normally **tip** displays verbose messages as it dials. See **remote**(5) for details of the remote host specification.

Depending on the type of auto-dialer being used to establish a connection the remote host may have garbage characters sent to it upon connection. The user should never assume that the first characters typed to the foreign host are the first ones presented to it. The recommended practice is to immediately type a **kill** character upon establishing a connection (most UNIX systems either support @ or CTRL-U as the initial kill character).

**tip** currently supports the Ventel MD-212+ modem and DC Hayes-compatible modems.

## REMOTE HOST DESCRIPTIONS

Descriptions of remote hosts are normally located in the system-wide file **/etc/remote**. However, a user may maintain personal description files (and phone numbers) by defining and exporting the REMOTE shell variable. The *remote* file must be readable by **tip**, but a secondary file describing phone numbers may be maintained readable only by the user. This secondary phone number file is **/etc/phones**, unless the shell variable PHONES is defined and exported. As described in **remote**(5), the *phones* file is read when the host description's phone number(s) capability is an '@'. The phone number file contains lines of the form:

*system-name phone-number*

Each phone number found for a system is tried until either a connection is established, or an end of file is reached. Phone numbers are constructed from '**0123456789**−=*', where the '=' and '*' are used to indicate a second dial tone should be waited for (ACU dependent).

## TIP INTERNAL VARIABLES

**tip** maintains a set of variables which are used in normal operation. Some of these variables are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the ~s escape. The syntax for variables is patterned after **vi**(1) and **mail**(1). Supplying **all** as an argument to the ~s escape displays all variables that the user can read. Alternatively, the user may request display of a particular variable by attaching a **?** to the end. For example '**~s escape?**' displays the current escape character.

Variables are numeric, string, character, or Boolean values. Boolean variables are set merely by specifying their name. They may be reset by prepending a **!** to the name. Other variable types are set by appending an = and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables.

Variables may be initialized at run time by placing set commands (without the ~s prefix) in a **.tiprc** file in one's home directory. The −v option makes **tip** display the sets as they are made. Comments preceded by a # sign can appear in the **.tiprc** file.

Finally, the variable names must either be completely specified or an abbreviation may be given. The following list details those variables known to **tip**.

**beautify**
(bool) Discard unprintable characters when a session is being scripted; abbreviated **be**. If the **nb** capability is present, **beautify** is initially set to *off*; otherwise, **beautify** is initially set to *on*.

**baudrate**
(num) The baud rate at which the connection was established; abbreviated **ba**. If a baud rate was specified on the command line, **baudrate** is initially set to the specified value; otherwise, if the **br** capability is present, **baudrate** is initially set to the value of that capability; otherwise, **baudrate** is set to 300 baud. Once **tip** has been started, **baudrate** can only changed by the super-user.

**dialtimeout**
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated **dial**. **dialtimeout** is initially set to 60 seconds, and can only changed by the super-user.

**disconnect**
(str) The string to send to the remote host to disconnect from it; abbreviated **di**. If the **di** capability is present, **disconnect** is initially set to the value of that capability; otherwise, **disconnect** is set to a null string ("").

**echocheck**
(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; abbreviated **ec**. If the **ec** capability is present, **echocheck** is initially set to *on*; otherwise, **echocheck** is initially set to *off*.

**eofread** (str) The set of characters which signify an EOT during a ~< file transfer command; abbreviated **eofr**. If the **ie** capability is present, **eofread** is initially set to the value of that capability; otherwise, **eofread** is set to a null string ("").

**eofwrite**
(str) The string sent to indicate EOT during a ~> file transfer command; abbreviated **eofw**. If the **oe** capability is present, **eofread** is initially set to the value of that capability; otherwise, **eofread** is set to a null string ("").

**eol** (str) The set of characters which indicate an end-of-line. **tip** will recognize escape characters only after an end-of-line. If the **el** capability is present, **eol** is initially set to the value of that capability; otherwise, **eol** is set to a null string ("").

**escape** (char) The command prefix (escape) character; abbreviated **es**. If the **es** capability is present, **escape** is initially set to the value of that capability; otherwise, **escape** is set to '~'.

**etimeout**
(num) The amount of time, in seconds, that **tip** should wait for the echo-check response when **echocheck** is set; abbreviated **et**. If the **et** capability is present, **etimeout** is initially set to the value of that capability; otherwise, **etimeout** is set to 10 seconds.

**exceptions**
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated **ex**. If the **ex** capability is present, **exceptions** is initially set to the value of that capability; otherwise, **exceptions** is set to '**\t\n\f\b**'.

**force** (char) The character used to force literal data transmission; abbreviated **fo**. If the **fo** capability is present, **force** is initially set to the value of that capability; otherwise, **force** is set to **\377** (which disables it).

**framesize**
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated **fr**. If the **fs** capability is present, **framesize** is initially set to the value of that capability; otherwise, **framesize** is set to 1024.

**halfduplex**
(bool) Do local echoing because the host is half-duplex; abbreviated **hdx**. If the **hd** capability is present, **halfduplex** is initially set to *on*; otherwise, **halfduplex** is initially set to *off*.

**host** (str) The name of the host to which you are connected; abbreviated **ho**. **host** is permanently set to the name given on the command line or in the HOST environment variable.

**localecho**
(bool) A synonym for **halfduplex**; abbreviated **le**.

**log** (str) The name of the file to which to log information about outgoing phone calls. **log** is initially set to **/var/adm/aculog**, and can only be inspected or changed by the super-user.

**parity** (str) The parity to be generated and checked when talking to the remote host; abbreviated **par**. The possible values are:

**none**
**zero** Parity is not checked on input, and the parity bit is set to zero on output.

**one** Parity is not checked on input, and the parity bit is set to one on output.

**even** Even parity is checked for on input and generated on output.

**odd** Odd parity is checked for on input and generated on output.

If the **pa** capability is present, **parity** is initially set to the value of that capability; otherwise, **parity** is set to **none**.

**phones** The file in which to find hidden phone numbers. If the environment variable PHONES is set, **phones** is set to the value of PHONES; otherwise, **phones** is set to **/etc/phones**. The value of **phones** cannot be changed from within **tip**.

**prompt** (char) The character which indicates an end-of-line on the remote host; abbreviated **pr**. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on receipt of this character. If the **pr** capability is present, **prompt** is initially set to the value of that capability; otherwise, **prompt** is set to **\n**.

**raise** (bool) Upper case mapping mode; abbreviated **ra**. When this mode is enabled, all lower case letters will be mapped to upper case by **tip** for transmission to the remote machine. If the **ra** capability is present, **raise** is initially set to *on*; otherwise, **raise** is initially set to *off*.

**raisechar**
(char) The input character used to toggle upper case mapping mode; abbreviated **rc**. If the **rc** capability is present, **raisechar** is initially set to the value of that capability; otherwise, **raisechar** is set to **\377** (which disables it).

**rawftp** (bool) Send all characters during file transfers; do not filter non-printable characters, and do not do translations like **\n** to **\r**. Abbreviated **raw**. If the **rw** capability is present, **rawftp** is initially set to *on*; otherwise, **rawftp** is initially set to *off*.

**record** (str) The name of the file in which a session script is recorded; abbreviated **rec**. If the **re** capability is present, **record** is initially set to the value of that capability; otherwise, **record** is set to **tip.record**.

**remote** The file in which to find descriptions of remote systems. If the environment variable REMOTE is set, **remote** is set to the value of REMOTE; otherwise, **remote** is set to **/etc/remote**. The value of **remote** cannot be changed from within **tip**.

**script** (bool) Session scripting mode; abbreviated **sc**. When **script** is *on*, **tip** will record everything transmitted by the remote machine in the script record file specified in **record**. If the **beautify** switch is on, only printable ASCII characters will be included in the script file (those characters between 040 and 0177). The variable **exceptions** is used to indicate characters which are an exception to the normal beautification rules. If the **sc** capability is present, **script** is initially set to *on*; otherwise, **script** is initially set to *off*.

**tabexpand**
(bool) Expand TAB characters to SPACE characters during file transfers; abbreviated **tab**. When **tabexpand** is *on*, each tab is expanded to 8 SPACE characters. If the **tb** capability is present, **tabexpand** is initially set to *on*; otherwise, **tabexpand** is initially set to *off*.

**tandem** (bool) Use XON/XOFF flow control to limit the rate that data is sent by the remote host; abbreviated **ta**. If the **nt** capability is present, **tandem** is initially set to *off*; otherwise, **tandem** is initially set to *on*.

**verbose** (bool) Verbose mode; abbreviated **verb**; When verbose mode is enabled, **tip** prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more. If the **nv** capability is present, **verbose** is initially set to *off*; otherwise, **verbose** is initially set to *on*.

**SHELL**
(str) The name of the shell to use for the ~! command; default value is **/bin/sh**, or taken from the environment.

**HOME** (str) The home directory to use for the ~c command; default value is taken from the environment.

**EXAMPLES**

An example of the dialogue used to transfer files is given below.

**arpa% tip monet**
**[connected]**
*...(assume we are talking to a UNIX system)..*
**ucbmonet login: sam**
**Password:**
**monet% cat > sylvester.c**
**~> Filename: sylvester.c**
**32 lines transferred in 1 minute 3 seconds**
**monet%**
**monet% ~< Filename: reply.c**
**List command for remote host: cat reply.c**
**65 lines transferred in 2 minutes**
**monet%**
*...(or, equivalently)..*
**monet% ~p sylvester.c**
*...(actually echoes as ~[put] sylvester.c)..*
**32 lines transferred in 1 minute 3 seconds**
**monet%**
**monet% ~t reply.c**
*...(actually echoes as ~[take] reply.c)..*
**65 lines transferred in 2 minutes**
**monet%**
*...(to print a file locally)..*
**monet% ~|Local command: pr –h sylvester.c | lpr**
**List command for remote host: cat sylvester.c**
**monet% ~^D**
**[EOT]**
*...(back on the local system)...*

**ENVIRONMENT**

The following environment variables are read by **tip**.

| | |
|---|---|
| **REMOTE** | The location of the *remote* file. |
| **PHONES** | The location of the file containing private phone numbers. |
| **HOST** | A default host to connect to. |
| **HOME** | One's log-in directory (for chdirs). |
| **SHELL** | The shell to fork on a '~!' escape. |

**FILES**

| | |
|---|---|
| **~/.tiprc** | initialization file |
| **/var/spool/uucp/LCK..*** | lock file to avoid conflicts with UUCP |
| **/var/adm/aculog** | file in which outgoing calls are logged |
| **/etc/phones** | |
| **/etc/remote** | |

**SEE ALSO**

**mail**(1), **uucp**(1C), **vi**(1), **ioctl**(2), **phones**(5), **remote**(5)

**BUGS**

There are two additional variables **chardelay** and **linedelay** that are currently not implemented.

NAME

toolplaces – display current SunView window locations, sizes, and other attributes

SYNOPSIS

**toolplaces** [ **–o|O|u** ] [ **–help** ]

AVAILABILITY

This command is available with the *SunView 1 User's* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**toolplaces** generates position, size, label, and program attributes for the windows running on a SunView screen at the time of execution. (**toolplaces** does not work when SunView is not running.)

Many people redirect standard output from **toolplaces** to the **.sunview** file, so as to reuse the current window system attributes each time they execute **sunview**(1).

For each window on the screen at execution time, **toolplaces** shows:

the tool name
the "open" window position
the size of the window in pixels
the "closed," or icon, window position
an indicator of whether the window is open or closed
the label at the top of the window
the name of the program running in the window, if a program is running there
any flags or options to a program running in the window

**toolplaces** describes each window on one output line, as long as necessary, using the current **sunview** format.

Current **sunview** format consists of window tool descriptions, one per line, as in this example (the \ indicates that the current line continues on the next line):

```
shelltool –Wp 491 795 –Ws 580 87 –WP 0 836 –C
clock –Wp 120 120 –Ws 122 55 –WP 1086 826 –Wi \
        –Wl " open clock" –S –r –d wdm
shelltool –Wp 491 166 –Ws 650 567 –WP 702 836 \
        –Wl due rlogin due
shelltool –Wp 0 0 –Ws 650 525 –WP 64 836 \
        –Wl "Small Window: /usr/bin/csh"
shelltool –Wp 501 0 –Ws 650 812 –WP 128 836 \
        –Wl "Big Window: /usr/bin/csh"
```

OPTIONS

**–o** Show window tool information in the old **suntools** format for window attributes, but specifies the appropriate tool names for each tool.

**–O** Show window tool information in the old **suntools** format for window attributes, specifying **toolname** as the name for each tool.

**–u** Show the updated window tool information in the order that you originally specified it.

**–help** Show help information preceding tool attributes.

FILES

**/.sunview** format file for **sunview**(1)

SEE ALSO

**sunview(1)**

*SunView 1 Beginner's Guide*

NAME

touch – update the access and modification times of a file

SYNOPSIS

**touch** [ **–c** ] [ **–f** ] *filename*...

SYSTEM V SYNOPSIS

**touch** [ **–c** ] [ **–a** ] [ **–m** ] [ *mmddhhmm*[*yy*] ] *filename*...

DESCRIPTION

**touch** sets the access and modification times of each argument to the current time. A file is created if it does not already exist.

**touch** is valuable when used in conjunction with **make**(1), where, for instance, you might want to force a complete rebuild of a program composed of many pieces. In such a case, you might type:

```
example% touch *.c
example% make
```

**make**(1) would then see that all the **.c** files were more recent than the corresponding **.o** files, and would start the compilation from scratch.

OPTIONS

**–c** Do not create *filename* if it does not exist.

**–f** Attempt to force the touch in spite of read and write permissions on *filename*.

SYSTEM V OPTIONS

**–a** Update only the access time.

**–m** Update only the modification time.

*mmddhhmm*[*yy*]

Update the times to the specified time rather than to the current time. The first *mm* is the month, **dd** is the day of the month, *hh* is the hour, and the second *mm* is the minute; if *yy* is specified, it is the last two digits of the year, otherwise the current year is used.

SEE ALSO

**make**(1), **utimes**(2)

BUGS

It is difficult to touch a file whose name consists entirely of digits in the System V **touch**, as it will interpret the first such non-flag argument as a time. You must ensure that there is a character in the name which is not a digit, by specifying it as *./name* rather than *name*.

NAME

tput – initialize a terminal or query the terminfo database

SYNOPSIS

**/usr/5bin/tput** [ –T*type* ] *capability* [ *parameter* ... ]

**/usr/5bin/tput** [ –T*type* ] **init**

**/usr/5bin/tput** [ –T*type* ] **longname**

**/usr/5bin/tput** [ –T*type* ] **reset**

AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**tput** uses the **terminfo**(5V) database to make the values of terminal-dependent capabilities and information available to the shell, to initialize or reset the terminal, or return the long name of the requested terminal type. Normally, the terminal type is taken from the environment variable TERM; if the –T*type* option is specified, the terminal type is taken from that option.

**tput** displays a string if the given *capability* is a string capability, or an integer if it is an integer capability; it displays no output if the capability is a boolean.

If *capability* is a boolean, **tput** returns true (0) if that capability is set, or false (1) otherwise. If *capability* is a string, **tput** returns true (0) if that capability is set, or false (1) otherwise. If *capability* is an integer, a value of true (0) is returned *whether or not* the capability is set for the terminal. To determine if an integer capability is set, you must examine the standard output.

For a complete list of capabilities and the *capability* associated with each, see **terminfo**(5V).

If *capability* is a string capability that takes parameters, the *parameter* arguments are instantiated into the string. An all-numeric *parameter* argument is passed to the attribute as a number.

OPTIONS

–T*type* Indicate the *type* of terminal. If this option is supplied, the environment variables LINES and COLUMNS are not used.

**init** If the **terminfo** database is present and an entry for the user's terminal exists, emit the terminal's initialization strings, set up any delays specified, and turn the expansion of TAB characters on or off, as specified by the terminal's entry in the **terminfo** database. If the terminal has a TAB character, and either has a capability for setting TAB characters or initially has its TAB characters set every 8 SPACE characters, expansion of TAB characters is turned off, otherwise expansion of TAB characters is turned on. If expansion of TAB characters is turned on, and the terminal has a capability for setting TAB characters, TAB stops are set to every eight columns. If an entry does not contain the information needed for any of these actions, that action is silently skipped.

**reset** Emit the terminal's reset strings, and set up delays and TAB characters as specified. If the reset strings are not present, but initialization strings are, the initialization strings are used.

**longname**

If the **terminfo** database is present and an entry for the user's terminal exists, emit the long name of the terminal. The long name is the last name in the first line of the terminal's description in the **terminfo** database.

EXIT CODES

0 The boolean or string capability is set, or the capability is an integer type and is present.

1 The *capability* is not set.

2 Usage error.

3 The terminal is of an unknown type, or the **terminfo** database is not present.

4 Unknown **terminfo** capability.

−1 The integer capability is not defined for this terminal type.

EXAMPLES

**tput init** Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command can be included in a **.profile** or **.login** file.

**tput −Tsun reset** Reset a Sun workstation console, **shelltool**(1) window, or **cmdtool**(1) window, overriding the type of terminal in the environmental variable **TERM**.

**tput cup 0 0** Send the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position).

**tput clear** Echo the clear-screen sequence for the current terminal.

**tput cols** Print the number of columns for the current terminal.

**tput −Tsun cols** Print the number of columns for the Sun workstation console or subwindow.

**bold='tput smso'**

**offbold='tput rmso'** Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt:

**echo "${bold}Please type in your name: ${offbold}\c"**

**tput hc** Set exit code to indicate if the current terminal is a hardcopy terminal.

**tput cup 23 4** Send the sequence to move the cursor to row 23, column 4.

**tput longname** Print the long name from the **terminfo** database for the type of terminal specified in the environmental variable **TERM**.

FILES

**/usr/share/lib/terminfo/?/*** compiled terminal description database

**/usr/5include/curses.h** **curses**(3X) header file

**/usr/5include/term.h** **terminfo**(5V) header file

**/usr/share/lib/tabset/*** TAB settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and TAB characters); for more information, see the **Tabs and Initialization** section of **terminfo**(5V).

**.login**

**.profile**

SEE ALSO

**cmdtool**(1), **shelltool**(1), **stty**(1V), **curses**(3X), **terminfo**(5V)

## NAME

tr – translate characters

## SYNOPSIS

**tr** [ **–cds** ] [ *string1* [ *string2* ] ]

## DESCRIPTION

**tr** copies the standard input to the standard output with substitution or deletion of selected characters. The arguments *string1* and *string2* are considered sets of characters. Any input character found in *string1* is mapped into the character in the corresponding position within *string2*. When *string2* is short, it is padded to the length of *string1* by duplicating its last character.

In either string the notation:

*a–b*

denotes a range of characters from *a* to *b* in increasing ASCII order. The character \, followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. As with the shell, the escape character \, followed by any other character, escapes any special meaning for that character.

## SYSTEM V DESCRIPTION

When *string2* is short, characters in *string1* with no corresponding character in *string2* are not translated.

In either string the following abbreviation conventions introduce ranges of characters or repeated characters into the strings. Note: in the System V version, square brackets are required to specify a range.

[*a–z*] Stands for the string of characters whose ASCII codes run from character *a* to character *z*, inclusive.

[*a***n*] Stands for *n* repetitions of *a*. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

## OPTIONS

Any combination of the options **–c**, **–d**, or **–s** may be used:

**–c** Complement the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal;

**–d** Delete all input characters in *string1*.

**–s** Squeeze all strings of repeated output characters that are in *string2* to single characters.

## EXAMPLE

The following example creates a list of all the words in *filename1* one per line in *filename2*, where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect '\' from the shell. 012 is the ASCII code for NEWLINE.

**tr –cs A–Za–z '\012'** <*filename1* >*filename2*

In the System V version, this would be specified as:

**tr –cs '[A–Z][a–z]' '[\012*]'** <*filename1* >*filename2*

## SEE ALSO

**ed**(1), **expand**(1), **ascii**(7)

## BUGS

Will not handle ASCII NUL in *string1* or *string2*. **tr** always deletes NUL from input.

NAME

trace – trace system calls and signals

SYNOPSIS

**trace** [ **–ct** ] [ **–o** *filename* ] *command*

**trace** [ **–ct** ] [ **–o** *filename* ] **–p** *pid*

DESCRIPTION

**trace** intercepts the system calls and signals of a process. The name of the system call, its arguments and result are listed on the standard output or on the file given as an argument to the **–o** option. The **–c** option can be used to get a quick summary of system call and signal counts instead of having a full trace.

Given a *command* it runs the command tracing all system calls until **exit**(2).

Given a process ID using the **–p** option, it ''attaches'' itself to the process and begins tracing. The trace may be terminated at any time by a keyboard interrupt signal (CTRL-C). **trace** will respond by detaching itself from the traced process leaving it to continue running.

Each line in the trace contains the system call name, followed by its arguments in parentheses and its result. Error returns (result = –1) have the error name and error message appended. Signals are printed as a signal name followed by the signal number. The **–t** option prefixes each line of the trace with the time of day. Arguments are printed according to their type. Structure pointers are always printed as hex addresses. Character pointers are dereferenced and printed as a quoted string. Non-printing characters in strings are represented by escape codes. Only the first 32 bytes of strings are printed; longer strings have two dots appended following the closing quote.

**The quick brown fox jumps over t ..**

Strings with more than 50% non-printing characters are assumed to contain binary data and are represented by a NULL string followed by two dots.

EXAMPLE

```
example% trace date
gettimeofday (0x21474, 0x2147c) = 0
gettimeofday (0x21474, 0) = 0
gettimeofday (0xefffc78, 0x214ac) = 0
ioctl (1, 0x40067408, 0xefffa10) = -1 ENOTTY (Inappropriate ioctl for device)
fstat (1, 0xefffa30) = 0
getpagesize () = 8192
brk (0x27640) = 0
close (0) = 0
Thu Dec  4 14:16:36 PST 1986
write (1, "Thu Dec  4 14:16:36 PST 1986\n", 29) = 29
close (1) = 0
close (2) = 0
exit (0) = ?
example%
```

SEE ALSO

**exit**(2), **ptrace**(2)

**BUGS**

Programs that use the *setuid* bit do not have effective user ID privileges while being traced.

Child processes of a traced process are not traced.

A traced process ignores SIGSTOP.

A traced process runs slowly.

NAME

traffic – SunView program to display Ethernet traffic

SYNOPSIS

**traffic** [ **–h** *host* ] [ **–s** *subwindows* ]

AVAILABILITY

This command is available when both the *Networking Tools and Programs* and the *SunView 1 User's* software options are installed. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**traffic** graphically displays ethernet traffic. It gets statistics from **etherd**(8C), running on machine *host*. The tool is divided into subwindows, each giving a different view of network traffic.

OPTIONS

**–h** *host* Specify a host from which to get statistics. The default value of *host* is the machine that **traffic** is running on.

**–s** *subwindows*

Specify the number of subwindows to display initially. The default value of *subwindows* is 1.

SUBWINDOWS

To the right of each subwindow is a panel that selects what the subwindow is viewing. When *Size* is checked, than the size distribution of packets is displayed. *Proto* is for protocol, *Src* is for source of packet, and *Dst* is for destination of packet. Since it is not possible to show all possible sources, when *Src* is selected, only the 8 highest sources are displayed (and similarly for *Dst* ).

For each of these choices, the distribution is displayed by a histogram. The panel above each subwindow controls characteristics of the histograms. At the left of the panel is a shaded square, corresponding to one of the two shades of bars in the histogram. You can switch the shade by either clicking on the square with the left button, or bringing up a menu over the square with the right mouse button. When the light colored square is visible, then the slider in the center of the panel controls how often the light colored bars are updated. When the dark square is visible, then the slider refers to the dark bars of the histogram. To the right of the slider is a choice of *Abs* versus *Rel*. This selects whether the height of the histogram is *Absolute* in packets per second, or *Relative* in percent of total packets on the ethernet. Next in the panel are three small horizontal bars. When selected (that is, when a check mark appears to the left of the three bars), a horizontal grid appears on the histogram. Finally the button marked *Delete Me* will delete the subwindow.

The right hand panel also has a choice for *Load* Load is represented as a strip chart, rather than a histogram. The maximum value of the graph represents a load of 100%, that is 10 megabits per second on the ethernet. When *Load* is selected, there is only one slider, and no *Rel* versus *Abs* choice.

At the very top of the tool is a panel that contains filters, as well as a *Split* button that splits the tool and creates a new subwindow, and a *Quit* button that exits the tool. The filters apply to all the subwindows. When a filter is selected, a check mark appears to the left of the word *Filter*. There can be more than one filter active at the same time. The meaning of each filter is as follows. *Src* is a host or net, which can be specified either by name or address (similarly for *Dst*). *Proto* is an ip protocol, and can either be a name (such as *udp*, *icmp*) or a number. *Lnth* is either a packet length, or a range of lengths separated by a dash.

SEE ALSO

**etherd**(8C)

BUGS

If multiple copies of **traffic** are using the same copy of *etherd* , and one of them invokes a filter, then all the copies of **traffic** will be filtered.

NAME

troff – typeset or format documents

SYNOPSIS

**troff** [ **–abfiqtwz** ] [ **–m***package* ] [ **–n***N* ] [ **–o***pagelist* ] [ **–p***N* ] [ **–r***aN* ] [ **–s***N* ] [ *filenames* ] . . .

AVAILABILITY

This command is available with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**troff** formats text in the *filename*s. For historical reasons, output goes to a CAT/4 phototypesetter attached to **/dev/cat**, but nobody uses a CAT/4 anymore. Ordinarily, postprocessing software converts output to a form that can be printed on newer typesetters or laser printers. Default font width tables correspond to Times Roman on PostScript™ printers. See also the **nroff**(1) manual page, which describes a formatter for typewriter-like devices.

Input to **troff** is expected to consist of text interspersed with formatting requests and macros. If no *filename* argument is present, **troff** reads standard input. A '–' as a *filename* argument indicates that standard input is to be read at that point in the list of input files; **troff** reads the files named ahead of the '–' in the arguments list, then text from the standard input, and then text from the files named after the '–'.

If the file **/etc/adm/tracct** is writable, **troff** keeps printer accounting records there. The integrity of that file may be secured by making **troff** a "set-user-ID" program (see **chmod**(1V) for details on the **setuid** permission bit.)

OPTIONS

Options may appear in any order, but they all must appear before the first *filename*.

**–a** Send a printable ASCII approximation of the results to the standard output.

**–i** Read the standard input after the input files are exhausted.

**–q** Disable echoing during a **.rd** request.

**–t** Direct output to the standard output instead of the printer. Since this output is non-ASCII it is generally redirected to **lpr –t**.

**–m***package*
Prepend the macro file **/usr/lib/tmac/tmac.***package* to the input *filename*s. (Note that most references to macro packages include the leading "**m**" as part of the name; the **man**(7) macro package resides in **/usr/lib/tmac/tmac.an**).

**–n***N* Number first generated page *N*.

**–o***list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

**–r***aN* Set register *a* (one-character) to *N*.

Some options of **troff** only apply if you have a CAT/4 typesetter attached to your system. These options remain present for backward compatibility. However, this version of **troff** does not support this typesetter by default.

**–b** Report whether the typesetter is busy or available. No text processing is done.

**–f** Refrain from feeding paper out and stopping at the end of the print job on the typesetter.

**–w** Wait until typesetter is available, if currently busy.

**–z** Suppress all formatted output. Display only terminal messages produced by **.tm** requests and diagnostics.

**–pN** Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce elasped time on the typesetter.

**–s***N* Stop the phototypesetter every *N* pages. **troff** produces a trailer so you can change cassettes; resume by pressing the typesetter's start button.

FILES

| | |
|---|---|
| **/tmp/ta*** | temporary file |
| **/usr/lib/tmac/tmac.*** | standard macro files |
| **/usr/lib/term/*** | terminal driving tables for **nroff** |
| **/usr/lib/font/*** | font width tables for alternate mounted **troff** fonts |
| **/dev/cat** | phototypesetter |
| **/etc/adm/tracct** | accounting statistics for **/dev/cat** |

SEE ALSO

**checknr**(1), **chmod**(1V), **eqn**(1), **lpr**(1), **nroff**(1), **tbl**(1), **col**(1V), **printcap**(5), **man**(7), **me**(7), **ms**(7), **lpd**(8)

*Formatting Documents*

*Using* `nroff` *and* `troff`

DIAGNOSTICS

**No /dev/cat: try –t or –a**

The CAT/4 typesetter is not accessible from your machine. Combine the **–t** option of **troff** with the **–t** option of **lpr**(1) to get output on a laser printer or typesetter. For information on how to inform **lpd**(8) of a PostScript printer attached to a remote host, see **printcap**(5).

NAME

true, false – provide truth values

SYNOPSIS

**true**

**false**

DESCRIPTION

**true** and **false** are usually used in a Bourne shell script. They test for the appropriate status "true" or "false" before running (or failing to run) a list of commands.

EXAMPLE

The following Bourne shell script will be executed while the case status is "true".

```
while true
do
        command list
done
```

SEE ALSO

**csh**(1), **sh**(1)

DIAGNOSTICS

**true** has exit status zero.

NAME

tset, reset – establish or restore terminal characteristics

SYNOPSIS

**tset** [ **–InQrsS** ] [ **–e***c* ] [ **–k***c* ] [ **–m** [*port*–ID [ *baudrate*] : *type* ] ... ] [*type*]

**reset** [ – ] [ **–e***c* ] [ **–I** ] [ **–k***c* ] [ **–n** ] [ **–Q** ] [ **–r** ] [ **–s** ] [ **–S** ]
**–m** [ *indent* ] [ *test baudrate* ]: *type* ] ... [ *type* ]

DESCRIPTION

**tset** sets up your terminal, typically when you first log in. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, sending any sequences needed to properly initialized the terminal, and the like. **tset** first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each port is specified in the **/etc/ttytab** database. Type names for terminals may be found in the **termcap**(5) database. If a port is not wired permanently to a specific terminal (not hardwired) it is given an appropriate generic identifier such as **dialup**.

**reset** clears the terminal settings by turning off CBREAK and RAW modes, output delays and parity checking, turns on NEWLINE translation, echo and TAB expansion, and restores undefined special characters to their default state. It then sets the modes as usual, based on the terminal type (which will probably override some of the above). (See **stty**(1V) for more information.) All arguments to **tset** may be used with **reset**. **reset** also uses the **rs**= and **rf**= (reset string and file) instead of the initialization string and file from **/etc/termcap** . This is useful after a program dies and leaves the terminal in a funny state. Often in this situation, characters will not echo as you type them. You may have to type '**<LINEFEED>reset<LINEFEED>**' since '**<RETURN>**' may not work.

When no arguments are specified, **tset** reads the terminal type from the **TERM** environment variable and re-initializes the terminal, and performs initialization of mode, environment and other options at login time to determine the terminal type and set up terminal modes.

When used in a startup script (**.profile** for **sh**(1) users or **.login** for **csh**(1) users) it is desirable to give information about the type of terminal you will usually use on ports that are not hardwired. These ports are identified in **/etc/ttyab** as **dialup** or **plugboard**, etc. Any of the alternate generic names given in **/etc/termcap** may be used for the identifier. Refer to the **–m** option under OPTIONS for more information. If no mapping applies and a final *type* option, not preceded by a **–m**, is given on the command line then that type is used; otherwise the type found in the **/etc/ttytab** database is used as the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by **tset**, and information about the terminal's capabilities, to a shell's environment. This can be done using the **–**, **–s**, or **–S** options. (Refer to OPTIONS for more information.)

For the Bourne shell, put this command in your **.profile** file:

**eval `tset –s** *options...* **`**

or using the C shell, put this command in your **.login** file:

**eval `tset –s** *options...* **`**

With the C shell, it is also convenient to make an alias in your **.cshrc** file:

**alias tset 'eval `tset –s \!*`'**

This also allows the command:

**tset 2621**

to be invoked at any time to set the terminal and environment. *Note to Bourne Shell users:* It is *not* possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. If a process could set its parent's environment, none of this nonsense would be necessary in the first place.

Once the terminal type is known, **tset** sets the terminal driver mode. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. TAB and NEWLINE expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is '#', the erase character is changed as if -e had been used.

OPTIONS

**–** The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the **TERM** environment variable.

**–e***c* Set the erase character to be the named character *c* on all terminals. Default is the backspace key on the keyboard, usually **^H** (CTRL-H). The character *c* can either be typed directly, or entered using the circumflex-character notation used here.

**–i***c* Set the interrupt character to be the named character *c* on all terminals. Default is **^C** (CTRL-C). The character *c* can either be typed directly, or entered using the circumflex-character notation used here.

**–I** Suppress transmitting terminal-initialization strings.

**–k***c* Set the line kill character to be the named character *c* on all terminals. Default is **^U** (CTRL-U). The kill character is left alone if **–k** is not specified. Control characters can be specified by prefixing the alphabetical character with a circumflex (as in CTRL-U) instead of entering the actual control key itself. This allows you to specify control keys that are currently assigned.

**–n** Specify that the new tty driver modes should be initialized for this terminal. Probably useless since **stty new** is the default.

**–Q** Suppress printing the '**Erase set to**' and '**Kill set to**' messages.

**–r** In addition to other actions, reports the terminal type.

**–s** Output commands to set and export **TERM** and **TERMCAP**. This can be used with

```
set noglob
eval `tset –s ...
unset noglob
```

to bring the terminal information into the environment. Doing so makes programs such as **vi**(1) start up faster. If the **SHELL** environment variable ends with **csh**, C shell commands are output, otherwise Bourne shell commands are output.

**–S** Similar to the **–s** option, but produces two strings containing suitable values for the (environment) variables **TERM** and **TERMCAP**, respectively, and can be used as follows:

```
set noglob
set t=(`tset –S ...`)
setenv TERM $t[1]
setenv TERMCAP "$t[2]"
unset t
unset noglob
```

Since **–s** loads these values, its use is preferred. If the **SHELL** environment variable does not end with **csh**, **–S** produces the same Bourne shell commands that **–s** does.

**–m** [*port*-ID[*baudrate*]:*type*] ...

Specify (map) a terminal type when connected to a generic port (such as *dialup* or *plugboard*) identified by *port-ID*. The *baudrate* argument can be used to check the baudrate of the port and set the terminal type accordingly. The target rate is prefixed by any combination of the following operators to specify the conditions under which the mapping is made:

| | |
|---|---|
| **>** | Greater than |
| **@** | Equals or "at" |
| **<** | Less than |
| **!** | It is not the case that (negates the above operators) |
| **?** | Prompt for the terminal type. If no response is given, then *type* is selected by default. |

In the following example, the terminal type is set to **adm3a** if the port is a dialup with a speed of greater than 300 or to *dw2* if the port is a dialup at 300 baud or less. In the third case, the question mark preceding the terminal type indicates that the user is to verify the type desired. A NULL response indicates that the named type is correct. Otherwise, the user's response is taken to be the type desired.

**tset –m 'dialup>300:adm3a' –m 'dialup:dw2' –m 'plugboard:?adm3a'**

To prevent interpretation as metacharacters, the entire argument to **–m** should be enclosed in single quotes. When using the C shell, exclamation points should be preceded by a backslash (\).

EXAMPLES

These examples all use the '–' option. A typical use of **tset** in a **.profile** or **.login** will also use the **–e** and **–k** options, and often the **–n** or **–Q** options as well. These options have been omitted here to keep the examples short.

To select a 2621, you might put the following sequence of commands in your **.login** file (or **.profile** for Bourne shell users).

```
set noglob
eval `tset –s 2621`
unset noglob
```

If you have an h19 at home which you dial up on, but your office terminal is hardwired and known in **/etc/ttytab**, you might use:

```
set noglob
eval `tset –s –m dialup:h19`
unset noglob
```

If you have a switch which connects to various ports (making it impractical to identify which port you may be connected to), and use various terminals from time to time, you can select from among those terminals according to the *speed* or baud rate. In the example below, **tset** will prompt you for a terminal type if the baud rate is greater than 1200 (say, 9600 for a terminal connected by an RS-232 line), and use a Wyse 50 by default. If the baud rate is less than or equal to 1200, it will select a 2621. Note the placement of the question mark, and the quotes to protect the **>** and **?** from interpretation by the shell.

```
set noglob
eval `tset –s –m 'switch>1200:?wy' –m 'switch<=1200:2621'`
unset noglob
```

All of the above entries will fall back on the terminal type specified in **/etc/ttytab** if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals, and the terminal you use most often is an adm3a.

```
set noglob
eval `tset –s ?adm3a`
unset noglob
```

If the file **/etc/ttytab** is not properly set up and you want to make the selection based only on the baud rate, you might use the following:

```
set noglob
eval `tset -s -m '>1200:wy' 2621`
unset noglob
```

Here is a fancy example to illustrate the power of **tset** and to hopelessly confuse anyone who has made it this far. It quietly sets the erase character to BACKSPACE, and kill to CTRL-U. If the port is switched, it selects a Concept 100 for speeds less than or equal to 1200, and asks for the terminal type otherwise (the default in this case is a Wyse 50). If the port is a direct dialup, it selects Concept 100 as the terminal type. If logging in over the ARPANET, the terminal type selected is a Datamedia 2500 terminal or emulator. (Note the backslash escaping the NEWLINE at the end of the first line in the example.)

```
set noglob
eval `tset -e -k^U -Q -s -m 'switch<=1200:concept100' -m \          'switch:?wy' -m
dialup:concept100 -m arpanet:dm2500`
unset noglob
```

FILES

| | |
|---|---|
| **/etc/ttytab** | port name to terminal type mapping database |
| **/etc/termcap** | terminal capability database |
| **/usr/share/lib/tabset/*** | TAB setting sequences for various terminals. Pointed to by **termcap** entries. |
| **.login** | |
| **.profile** | |

SEE ALSO

**csh**(1), **sh**(1), **vi**(1), **stty**(1V), **ttytab**(5), **termcap**(5), **environ**(5V)

BUGS

The **tset** command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the **login** program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

This program cannot intuit personal choices for erase, interrupt and line kill characters, so it leaves these set to the local system standards.

It could well be argued that the shell should be responsible for ensuring that the terminal remains in a sane state; this would eliminate the need for the **reset** program.

**NAME**

tsort – topological sort

**SYNOPSIS**

**tsort** [ *filename* ]

**DESCRIPTION**

**tsort** produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *filename*. If no *filename* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by SPACE characters. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

**SEE ALSO**

**lorder**(1)

**BUGS**

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

**NAME**

tty – display the name of the terminal

**SYNOPSIS**

**tty** [ –s ]

**DESCRIPTION**

**tty** prints the pathname of the user's terminal unless the –s (silent) option is given. In either case, the exit value is zero if the standard input is a terminal, and one if it is not.

**OPTIONS**

–s Silent. Does not print the pathname of the user's terminal.

## NAME

ul – do underlining

## SYNOPSIS

**ul** [ **–i** ] [ **–t** *terminal* ] [ *filename*... ]

## DESCRIPTION

**ul** reads the named *filename*s (or the standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable **TERM**. **ul** uses the **/etc/termcap** file to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, **ul** degenerates to **cat**(1V). If the terminal cannot underline, underlining is ignored.

## OPTIONS

**–t** *terminal*

Override the terminal kind specified in the environment. If the terminal cannot underline, underlining is ignored.

**–i** Indicate underlining by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an **nroff**(1) output stream on a CRT-terminal.

## FILES

**/etc/termcap**

## SEE ALSO

**cat**(1V), **colcrt**(1), **man**(1), **nroff**(1)

## BUGS

**nroff** usually generates a series of backspaces and underlines intermixed with the text to indicate underlining. **ul** makes attempt to optimize the backward motion.

## NAME

uname – display the name of the current system

## SYNOPSIS

**uname** [ **–mnrsva** ]

## DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command. **uname** prints the current system name of the system on the standard output.

## OPTIONS

**–m** Print the machine hardware name.

**–n** Print the nodename (the nodename may be a name that the system is known by to a communications network).

**–r** Print the operating system release.

**–s** Print the system name (default).

**–v** Print the operating system version.

**–a** Print all the above information.

## SEE ALSO

**uname**(2V)

## NAME

unget – undo a previous get of an SCCS file

## SYNOPSIS

**/usr/sccs/unget** [ **–ns** ] [ **–r***SID* ] *filename*...

## DESCRIPTION

Unget undoes the effect of a '**get –e**' done prior to creating the intended new delta. If a directory is named, **unget** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of '–' is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

## OPTIONS

Options apply independently to each named file.

**–n** Retain the gotten file '–'itisnormally

**–s** Suppress displaying the intended delta's *SID*.

**–r***SID* Uniquely identify which delta is no longer intended. This would have been specified by **get** as the "new delta"). The **–r** option is necessary only if two or more outstanding gets for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary but omitted from the command line.

## SEE ALSO

**delta**(1), **get**(1), **help**(1), **sact**(1), **sccs**(1)

## DIAGNOSTICS

Use **help**(1) for explanations.

NAME

unifdef – resolve and remove ifdef'ed lines from cpp input

SYNOPSIS

**unifdef** [ **–c l t** ] [ **–D***name* ] [ **–U***name* ] [ **–id***name* ] [ **–iu***name* ] ... [ *filename* ]

DESCRIPTION

**unifdef** removes **ifdef**ed lines from a file while otherwise leaving the file alone. It is smart enough to deal with the nested **ifdef**s, comments, single and double quotes of C syntax, but it does not do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined with **–D** options, and which you want undefined with **–U** options. Lines within those **ifdef**s will be copied to the output, or removed, as appropriate. Any **ifdef**, **ifndef**, **else**, and **endif** lines associated with *filename* will also be removed.

**ifdef**s involving symbols you do not specify are untouched and copied out along with their associated **ifdef**, **else**, and **endif** lines.

If an **ifdef***X* occurs nested inside another **ifdef***X*, then the inside **ifdef** is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

**unifdef** copies its output to the standard output and will take its input from the standard input if no *filename* argument is given.

OPTIONS

**–c** Complement the normal operation. Lines that would have been removed or blanked are retained, and vice versa.

**–l** Replace "lines removed" lines with blank lines.

**–t** Plain text option. **unifdef** refrains from attempting to recognize comments and single and double quotes.

**–id***name*

Ignore, but print out, lines associated with the defined symbol *filename*. If you use **ifdef**s to delimit non-C lines, such as comments or code which is under construction, then you must tell **unifdef** which symbols are used for that purpose so that it won't try to parse for quotes and comments within them.

**–iu***name*

Ignore, but print out, lines associated with the undefined symbol *filename*.

SEE ALSO

**cpp**(1), **diff**(1)

DIAGNOSTICS

**Premature EOF**

Inappropriate **else** or **endif**.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

BUGS

Does not know how to deal with **cpp**(1) constructs such as

```
#if     defined(X) || defined(Y)
```

## NAME

uniq – remove or report adjacent duplicate lines

## SYNOPSIS

**uniq** [ **–cdu** [ +|*–n* ] [ *inputfile* [ *outputfile* ] ]

## DESCRIPTION

**uniq** reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note: repeated lines must be adjacent in order to be found; see **sort**(1V).

## OPTIONS

**–c** Supersede **–u** and **–d** and generate an output report in default style but with each line preceded by a count of the number of times it occurred.

The normal output of **uniq** is the union of the **–u** and **–d** options.

**–d** Write one copy of just the repeated lines.

**–u** Copy only those lines which are *not* repeated in the original file.

The *n* arguments specify skipping an initial portion of each line in the comparison:

*+n* The first *n* characters are ignored. Fields are skipped before characters.

*–n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-SPACE, non-TAB characters separated by SPACE and TAB characters from its neighbors.

## SEE ALSO

**comm**(1), **sort**(1V)

## NAME

units – conversion program

## SYNOPSIS

**units**

## DESCRIPTION

**units** converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

**units** only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

| | |
|---|---|
| **pi** | Ratio of circumference to diameter, |
| **c** | Speed of light, |
| **e** | Charge on an electron, |
| **g** | Acceleration of gravity, |
| **force** | Same as g, |
| **mole** | Avogadro's number, |
| **water** | Pressure head per unit height of water, |
| **au** | Astronomical unit. |

**pound** is not recognized as a unit of mass; **lb** is. **pound** refers to a British pound. Compound names are run together (for instance, **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. Currency is denoted **belgiumfranc**, **britainpound**, ... For a complete list of units, type:

**cat /usr/lib/units**

## FILES

**/usr/lib/units**

## BUGS

Do not base your financial plans on the currency conversions.

NAME

unix2dos – convert text file from SunOS format to DOS format

SYNOPSIS

**unix2dos** [ **–iso** ] [ **–7** ] *originalfile convertedfile*

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**unix2dos** adds carriage returns and converts end of file characters in SunOS format text files to conform to DOS requirements.

This command may be invoked from either DOS or SunOS. However, the filenames must conform to the conventions of the environment in which the command is invoked.

If the original file and the converted file are the same, **unix2dos** will rewrite the original file after converting it.

OPTIONS

**–iso** Convert ISO standard characters to the corresponding character in the DOS extended character set.

**–7** Convert 8 bit SunOS characters to 7 bit DOS characters.

DIAGNOSTICS

**File** *filename* **not found, or no read permission**

The input file you specified does not exist, or you do not have read permission (check with the SunOS command ls -l).

**Bad output filename** *filename*, **or no write permission**

The output file you specified is either invalid, or you do not have write permission for that file or the directory that contains it. Check also that the drive or diskette is not write-protected.

**Error while writing to temporary file**

An error occurred while converting your file, possibly because there is not enough space on the current drive. Check the amount of space on the current drive using the DIR command. Also be certain that the default diskette or drive is write-enabled (not write-protected). Note that when this error occurs, the original file remains intact.

**Could not rename tmpfile to** *filename*.
**Translated tmpfile name =** *filename*.

The program could not perform the final step in converting your file. Your converted file is stored under the name indicated on the second line of this message.

SEE ALSO

*Sun386i Advanced Skills*
*DOS Reference Manual*

NAME

unload, unloadc – unload Application SunOS or Developer's Toolkit optional clusters

SYNOPSIS

**unload** *filename*...

**unloadc** *cluster*...

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**unload** unloads the Application SunOS or Developer's Toolkit clusters that contain the specified file arguments. **unloadc** unloads the specified Application SunOS or Developer's Toolkit clusters.

Without arguments, **unload** and **unloadc** display a summary of the clusters in the Application SunOS and Developer's Toolkit, including the load state and size of each cluster.

EXAMPLES

To unload the cluster that contains the **spell** command:

```
example% unload spell
About to unload the spellcheck cluster, confirm (y/n): y
Unloading the spellcheck cluster ...
The spellcheck cluster has been unloaded.
space used by clusters: 5358K bytes
total space remaining: 20962K bytes
example%
```

To display a summary of the clusters in the Application SunOS and Developer's Toolkit:

```
example% unload
Application SunOS Clusters:
  available  cluster          size (bytes)
  ---------  -------          ----
    yes      accounting       265K
    no       advanced_admin   501K
    ...

Developer's Toolkit Clusters:
  available  cluster          size (bytes)
  ---------  -------          ----
    no       base_devel       6907K
    ...

space used by clusters: 6021K bytes
total space remaining: 20432K bytes
```

FILES

**/export/loaded/appl**
where Application SunOS clusters are loaded (or mounted)

**/export/loaded/devel**
where Developer's Toolkit clusters are loaded (or mounted)

**/usr/lib/load/***
data files

SEE ALSO

**cluster**(1), **load**(1), **toc**(5)
**Sun386i Setup and Maintenance**

DIAGNOSTICS

**The file *filename* is not in any of the optional software clusters.**

The specified file is not part of the Application SunOS or Developer's Toolkit.

**The cluster *cluster* is not loaded.**

The specified cluster is not loaded on disk.

**There is no *cluster* cluster.**

The specified cluster is not part of the Application SunOS or Developer's Toolkit.

**The Application SunOS (and/or) Developer's Toolkit are mounted.**

The Application SunOS or Developer's Toolkit or both are mounted across the network and can not be loaded or unloaded.

NAME

uptime – show how long the system has been up

SYNOPSIS

**uptime**

DESCRIPTION

**uptime** prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a w(1) command.

EXAMPLE

```
example% uptime
6:47am  up 6 days, 16:38,  1 users,  load average: 0.69, 0.28, 0.17
example%
```

FILES

**/vmunix** system name list

SEE ALSO

w(1)

NAME

users – display a compact list of users logged in

SYNOPSIS

**users**

DESCRIPTION

**users** lists the login names of the users currently on the system in a compact, one-line format:

```
example% users
paul george ringo
example%
```

FILES

**/etc/utmp**

SEE ALSO

**who(1)**

## NAME

uucp, uulog, uuname – system to system copy

## SYNOPSIS

**uucp** [ **–acCdfmr** ] [ **–e***system* ] [ **–n***username* ] [ **–g***grade* ] [ **–s***spool* ] [ **–x***debug* ] *source-file* ... *destination-file*

**uulog** [ **–s***system* ] [ **–u***username* ]

**uuname** [ **–l** ]

## AVAILABILITY

This command is available with the **uucp** software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**uucp** copies each *source-file* to the named *destination-file*. A filename may be a path name on your machine, or may have the form

*system-name*!*pathname*

where *system-name* is taken from a list of system names that **uucp** knows about. Shell metacharacters ?, *, and [ ] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of:

- a full pathname;
- a pathname preceded by *username*/; where *username* is a username on the specified system and is replaced by that user's login directory;
- a pathname preceded by /; such a pathname will be replaced by the "public uucp" directory on the remote machine;
- anything else is prefixed by the pathname of the current directory.

If the result is an erroneous pathname for the remote system, the copy will fail. If the *destination-file* is a directory, the last component of the *source-file* name is used.

**uucp** preserves execute permissions across the transmission and gives 0666 read and write permissions (see **chmod**(2)).

**uulog** maintains a summary log of **uucp** and **uux**(1C) transactions in the file **/var/spool/uucp/LOGFILE**, by gathering information from partial log files named **/var/spool/uucp/LOG.*.?**. It removes the partial log files.

**uuname** lists the **uucp** names of systems that can be accessed using **uucp.**

## OPTIONS

### uucp Options

**–a** Avoid doing a **getwd**(3) to find the current directory. This is sometimes used for efficiency.

**–c** Use the source file when copying out rather than copying the file to the spool directory. This is the default.

**–C** Make a copy of outgoing files in the **uucp** spool directory, rather than copying the source file directly to the target system. This lets you remove the source file after issuing the **uucp** command.

**–d** Make all necessary directories for the file copy.

**–f** Do not make intermediate directories for the file copy.

**–m** Send mail to the requester when the copy is complete.

**–r** Do not start the transfer, just queue the job.

**–e***system*
Send the **uucp** command to the system *system* to be executed there. This works only when the remote machine allows **uucp** to be executed by **/usr/lib/uucp/uuxqt**.

**–n***username*
Notify *username* on remote system (by mail) that a file was sent.

**–g***grade*
*grade* is a single letter or number; lower ASCII values transmit a job earlier during a particular conversation. The default *grade* is **n**. By way of comparison, **uux**(1C) defaults to '**A**'; mail is usually sent at grade '**C**'.

**–s***spool* Use *spool* as the spool directory instead of the default.

**–x***debug*
Turn on the debugging at level *debug*.

**uulog Options**

**–s***system*
Print information about work involving system *system*.

**–u***username*
Print information about work done for the specified *username*.

**uuname options**

**–l** Display the local system-name.

## FILES

| | |
|---|---|
| **/var/spool/uucp** | spool directory |
| **/usr/lib/uucp/ADMIN** | list of known systems and descriptions |
| **/usr/lib/uucp/*** | other data and program files |
| **/var/spool/uucp/LOGFILE** | |

## SEE ALSO

**mail**(1), **uux**(1C), **chmod**(2), **getwd**(3)

*System and Network Administration*

## WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

## BUGS

All files received by **uucp** will be owned by the user ID **uucp**.

The **–m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **?**, *, and [ ] will not activate the **–m** option.

## NAME

uuencode, uudecode – encode a binary file, or decode its ASCII representation

## SYNOPSIS

**uuencode** [ *source-file* ] *file-label*

**uudecode** [ *encoded-file* ]

## DESCRIPTION

**uuencode** converts a binary file into an ASCII-encoded representation that can be sent using **mail**(1). It encodes the contents of *source-file*, or the standard input if no *source-file* argument is given. The *file-label* argument is required. It is included in the encoded file's header as the name of the file into which **uudecode** is to place the binary (decoded) data. **uuencode** also includes the ownership and permission modes of *source-file*, so that *file-label* is recreated with those same ownership and permission modes.

If the remote host is a UNIX system with the **sendmail**(8) mail-message delivery daemon, you can pipe the output of **uuencode** through **mail**(1) to the recipient named **decode** on the remote host. This recipient is typically an alias for the **uudecode** program (see **aliases**(5) for details), which allows a binary file to be decoded (extracted) from a mail message automatically. If this alias is absent on a particular host, the encoded file can be mailed to a user, who can run it through **uudecode** manually.

*uudecode* reads an *encoded-file*, strips off any leading and trailing lines added by mailer programs, and recreates the original binary data with the filename and the mode and owner specified in the header.

The encoded file is an ordinary ASCII text file; it can be edited by any text editor. But it is best only to change the mode or file-label in the header to avoid corrupting the decoded binary.

## SEE ALSO

**mail**(1), **uucp**(1C), **uusend**(1C), **uux**(1C), **aliases**(5), **uuencode**(5), **sendmail**(8)

## BUGS

The encoded file's size is expanded by 35% (3 bytes become 4, plus control information), causing it to take longer to transmit than the equivalent binary.

The user on the remote system who is invoking **uudecode** (typically **uucp**) must have write permission on the file specified in the *file-label*.

Since both **uuencode** and **uudecode** run with user ID set to **uucp**, **uudecode** can fail with "permission denied" when attempted in a directory that does not have write permission allowed for "other."

## NAME

uusend – send a file to a remote host

## SYNOPSIS

**uusend** [ **–m** *mode* ] *sourcefile sys1*!*sys2*!...!*remotefile*

## AVAILABILITY

This command is available with the **uucp** software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**uusend** sends a file to a given location on a remote system. The system need not be directly connected to the local system, but a chain of **uucp**(1C) links needs to connect the two systems.

The sourcefile can be '–', meaning to use the standard input. Both of these options are primarily intended for internal use of **uusend**.

The remotefile can include the *username* or / syntax.

## OPTIONS

**–m** *mode*

Take the mode of the file on the remote end from the octal number specified as *mode*. The mode of the input file is used if the **–m** option is not specified.

## SEE ALSO

**uucp**(1C), **uuencode**(1C), **uux**(1C)

## BUGS

This command should not exist, since **uucp** should handle it.

All systems along the line must have the **uusend** command available and allow remote execution of it.

Some UUCP systems have a bug where binary files cannot be the input to a **uux** command. If this bug exists in any system along the line, the file will show up severely corrupted.

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

**uustat –a|–m|–p| |–k***jobid* ] |–r*jobid* ]

**uustat** [ –s*system* ] [ –u*user* ]

AVAILABILITY

This command is available with the software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**uustat** displays the status of, or cancels, previously specified **uucp**(1C) commands. It also reports the status of **uucp** connections to other systems. When no options are given, **uustat** displays the status of all **uucp** requests issued by the current user.

OPTIONS

Only one of the following options can be specified at a time:

**–a** Output all jobs in queue.

**–m** Report the status of accessibility of all machines.

**–p** Execute a **ps –flp** for all the PIDs listed in the lock files.

**–q** List the jobs queued for each machine. If a status file exists for the machine, its date, time status information are reported. In addition, if a number appears in parentheses next to the number of C or X files, it is the age in days of the oldest **C./X.** file for that system. The **Retry** field represents the number of hours until the next possible call. The **Count** is the number of failure attempts. For systems with a moderate number of outstanding jobs, this could take 30 seconds or more to execute. An example of the output from **–q** is:

**eagle 3C 04/07–11:07NO DEVICES AVAILABLE**
**mh3bs3 2C 07/07–10:42SUCCESSFUL**

This indicates the number of command files that are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

**–k***jobid* Kill the **uucp** request with job identification of *jobid*. You must either own the job to be killed, or be the super-user.

**–r***jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the next limit imposed by the daemon.

The following options can be specified separately or together:

**–s***sys* Report the status of all **uucp** requests for remote system *sys*.

**–u***user* Report the status of all **uucp** requests issued by user.

Output for both the **–s** and **–u** options has the following format:

**eaglen0000 4/07–11:01:03(POLL)**
**eagleN1bd7 4/07–11:07Seagledan522 /usr/dan/A**
**eagleC1bd8 4/07–11:07Seagledan59 D.3b2al2ce4924**
**4/07–11:07Seagledanrmail mike**

The first field is the job ID. This is followed by the date and time. The next field is either an **S** or **R** depending on whether the job is to send or request a file. This is followed by the user ID of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution request, the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user, or an internal name created for data files associated with remote executions (**rmail** in this example).

**FILES**

**/var/spool/uucp/*** **uucp** spool directories

**SEE ALSO**

**uucp**(1C)

## NAME

uux – remote system command execution

## SYNOPSIS

**uux** [ – ] [ **–nrz** ] [ **–g***x* ] [ **–x***n* ] *command-string*

## AVAILABILITY

This command is available with the **uucp** software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**uux** will gather 0 or more files from various systems, execute a command on a specified system and send the standard output to a file on a specified system.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by '*system-name*!'. A null system-name is interpreted as the local system.

File names may be one of:

- a full pathname;
- a pathname preceded by *xxx*/; where *xxx* is a username on the specified system and is replaced by that user's login directory;
- a pathname preceded by '~/'; such a pathname is replaced by the "public uucp" directory on the remote machine;
- anything else is prefixed by the current directory.

The '–' option sends the standard input to the **uux** command as the standard input to the command-string.

Any special shell characters such as <>, ;, and | should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

## OPTIONS

**–n** Do not return any indication by **mail**(1) of success or failure of the job.

**–r** Do not start **uucico**, just queue the job.

**–z** Return an indication by **mail** only if the job fails.

**–g***x* Set service grade or classification to *x*.*The* default*is* **A**.

**–x***n* Set debugging level to *n*. (5, 7, and 9 are good numbers to try; they give increasing amounts of detail.)

## EXAMPLE

The command

**uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !fi.diff"**

will get the **f1** files from the **usg** and **pwba** machines, execute a **diff** command and put the results in **f1.diff** in the local directory.

## FILES

| | |
|---|---|
| **/var/spool/uucp** | spool directory |
| **/usr/lib/uucp/*** | other data and programs |

## SEE ALSO

**mail**(1), **uucp**(1C), **sendmail**(8)

*System and Network Administration*

**WARNING**

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an incoming request from **uux**. Typically, a restricted site will permit little other than the receipt of mail using **uux**.

**BUGS**

Only the first command of a shell pipeline may have a '*system-name*!'. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do.

The shell tokens << and >> are not implemented.

There is no notification of denial of execution on the remote machine.

NAME

vacation – reply to mail automatically

SYNOPSIS

**vacation** [ **–I** ]
**vacation** [ **–j** ] [ **–a***alias* ] [ **–t***N* ] *username*

DESCRIPTION

**vacation** automatically replies to incoming mail. The reply is contained in the file **.vacation.msg**, that you create in your home directory.

This file should include a header with at least a '**Subject:**' line (it should not include a '**From:**' or a '**To:**' line). For example:

**Subject: I am on vacation**
**I am on vacation until July 22. If you have something urgent,**
**please contact Joe Jones (jones@f40).**
**--John**

If the string **$SUBJECT** appears in the **.vacation.msg** file, it is replaced with the subject of the original message when the reply is sent; thus, a **.vacation.msg** file such as

**Subject: I am on vacation**
**I am on vacation until July 22.**
**Your mail regarding "$SUBJECT" will be read when I return.**
**If you have something urgent, please contact Joe Jones (jones@f40).**
**--John**

will include the subject of the message in the reply.

No message is sent if the '**To:**' or the '**Cc:**' line does not list the user to whom the original message was sent or one of a number of aliases for them, if the initial **From** line includes the string **–REQUEST@**, or if a '**Precedence: bulk**' or '**Precedence: junk**' line is included in the header.

OPTIONS

**–I** Initialize the **.vacation.pag** and **.vacation.dir** files and start **vacation**.

If the **–I** flag is not specified, and a *user* argument is given, **vacation** reads the first line from the standard input (for a '**From:**' line, no colon). If absent, it produces an error message. The following options may be specified:

**–a***alias* Indicate that *alias* is one of the valid aliases for the user running **vacation**, so that mail addressed to that alias generates a reply.

**–j** Do not check whether the recipient appears in the '**To:**' or the '**Cc:**' line.

**–t***N* Change the interval between repeat replies to the same sender. The default is 1 week. A trailing **s, m, h, d,** or **w** scales *N* to seconds, minutes, hours, days, or weeks respectively.

USAGE

To start **vacation**, create a **.forward** file in your home directory containing a line of the form:

*****username***, "|/usr/ucb/vacation** *username***"**

where *username* is your login name.

Then type in the command:

**vacation –I**

To stop **vacation**, remove the **.forward** file, or move it to a new name.

If **vacation** is run with no arguments, it will permit you to interactively turn **vacation** on or off. It will create a **.vacation.msg** file for you, or edit an existing one, using the editor specified by the VISUAL or EDITOR environment variable, or vi(1) if neither of those environment variables are set. If a **.forward** file is

present in your home directory, it will ask whether you want to remove it and turn off **vacation**. If it is not present in your home directory, it creates it for you, and automatically performs a '**vacation –I**' function, turning on **vacation**.

FILES

**.forward**
**$HOME/.vacation.mesg**

A list of senders is kept in the files **.vacation.pag** and **.vacation.dir** in your home directory.

SEE ALSO

**vi**(1), **sendmail**(8)

## NAME

val – validate an SCCS file

## SYNOPSIS

**/usr/sccs/val** –

**/usr/sccs/val** [ **–s** ] [ **–m** *name* ] [ **–r***SID* ] [ **–y** *type* ] *filename*...

## DESCRIPTION

**val** determines if the specified *filename*s are SCCS files meeting the characteristics specified by the optional argument list. Arguments to **val** may appear in any order. **val** can process up to 50 files on a single command line.

**val** has a special argument, '–', which reads the standard input until an EOF condition is detected. Each line read is independently processed as if it were a command line argument list.

**val** generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8–bit code upon exit as described below.

## OPTIONS

Options apply independently to each named file on the command line.

**–s** Silence diagnostic messages normally generated for errors detected while processing the specified files.

**–m** *name*
*filename* is compared with the SCCS **%M%** keyword in *filename*.

**–r** *SID* The argument value *SID* (SCCS ID String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (for instance, '**r 1**' is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (for instance, '**r 1.0**' or '**r 1.1.0**' are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

**–y** *type* *type* is compared with the SCCS **%Y%** keyword in *filename*.

The 8-bit code returned by **val** is a disjunction of the possible errors, that is, can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate option;
bit 2 = corrupted SCCS file;
bit 3 = can not open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = **%Y%**, **–y** mismatch;
bit 7 = **%M%**, **–m** mismatch;

Note: **val** can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned — logical **OR** of the codes generated for each command line and file processed.

## SEE ALSO

**admin**(1), **delta**(1), **get**(1), **help**(1), **prs**(1), **sccs**(1)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use **help**(1) for explanations.

NAME

vfontinfo – inspect and print out information about fonts

SYNOPSIS

**/usr/lib/vfontinfo** [ **–v** ] *fontname* [ *characters* ]

AVAILABILITY

This command is available with the *Versatec Printer* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**vfontinfo** allows you to examine a font in the UNIX system format. It prints out all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format.

The *fontname* argument is the name of the font you wish to inspect. It writes to the standard output. If it cannot find the file in your working directory, it looks in **/usr/lib/vfont** (the place most of the fonts are kept).

The *characters*, if given, specify certain characters to show. If omitted, the entire font is shown.

OPTIONS

**–v** Verbose. The bits of the glyph itself are shown as an array of X's and SPACE characters, in addition to the header information.

FILES

**/usr/lib/vfont**
**/usr/lib/vpd** Versatec daemon

SEE ALSO

**vswap**(1), **vwidth**(1), **vfont**(5)

## NAME

vgrind – grind nice program listings

## SYNOPSIS

**vgrind** [ – ] [ **–ftnxWw** ] [ **s***n* ] [ **–h** *header* ] [ **–d** *defs-file* ] [ **–l***language* ] *filename*...

## AVAILABILITY

This command is available for Sun-2, Sun-3 and Sun-4 systems with the *Text Processing Tools* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**vgrind** formats the program sources named by the *filename* arguments in a nice style using **troff**(1). Comments are placed in italics, keywords in bold face, and as each function is encountered its name is listed on the page margin.

**vgrind** runs in two basic modes, filter mode or regular mode. In filter mode **vgrind** acts as a filter in a manner similar to **tbl**(1). The standard input is passed directly to the standard output except for lines bracketed by the **troff**-like macros:

.v**S** starts processing

.v**E** ends processing

These lines are formatted as described above. The output from this filter can be passed to **troff** for output. There need be no particular ordering with **eqn**(1) or **tbl**.

In regular mode **vgrind** accepts input *filename*s, processes them, and passes them to **troff** for output.

In both modes **vgrind** passes any lines beginning with a decimal point without conversion.

## OPTIONS

Note: arguments to the –l and –s options follow the option names immediately, with no intervening space. All other arguments and options must be separated with white space.

– Take from standard input (default if **–f** is specified).

**–f** Force filter mode.

**–t** Similar to the same option in **troff**; that is, formatted text goes to the standard output.

**–n** Do not make keywords boldface.

**–x** Output the index file in a "pretty" format. The index file itself is produced whenever **vgrind** is run with a file called **index** present in the current directory. The index of function definitions can then be run off by giving **vgrind** the **–x** option and the file **index** as argument.

**–W** Force output to the (wide) Versatec printer rather than the (narrow) Varian.

**–w** Consider TAB characters to be spaced four columns apart instead of the usual eight.

**–s***n* Specify a point size to use on output (exactly the same as the argument of a **troff .ps** point size request).

**–h** *header*
Specify a particular header to put on every output page (default is the current file name).

**–d** *defs-file*
Specify an alternate language definitions file (default is **/usr/lib/vgrindefs**).

**–l***language*
Specify the language to use. Among the languages currently known are: Bourne shell (**–lsh**), C (**–lc**, the default), C-shell (**–lcsh**), emacs MLisp, (**–lml**), FORTRAN (**–lf**), Icon (**–ll**), ISP (**–i**), LDL (**–lLDL**), Model (**–lm**), Pascal (**–lp**), and RATFOR (**–lr**).

ENVIRONMENT

In regular mode **vgrind** feeds its intermediate output to the text formatter given by the value of the **TROFF** environment variable, or to **TROFF** if this variable is not defined in the environment. This mechanism allows for local variations in **troff**'s name.

FILES

| | |
|---|---|
| **index** | file where source for index is created |
| **/usr/lib/vgrindefs** | language descriptions |
| **/usr/lib/vfontedpr** | preprocessor |
| **/usr/share/lib/tmac/tmac.vgrind** | macro package |

SEE ALSO

**ctags**(1), **eqn**(1), **tbl**(1), **troff**(1), **vgrindefs**(5)

BUGS

**vgrind** assumes that a certain programming style is followed:

| | |
|---|---|
| C | Function names can be preceded on a line only by SPACE, TAB, or an asterisk. The parenthesized arguments must also be on the same line. |
| FORTRAN | Function names need to appear on the same line as the keywords *function* or *subroutine*. |
| MLisp | Function names should not appear on the same line as the preceding *defun*. |
| Model | Function names need to appear on the same line as the keywords *is beginproc*. |
| Pascal | Function names need to appear on the same line as the keywords *function* or *procedure*. |

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of spaces to align source code fails miserably; if you plan to **vgrind** your program you should use TAB characters. This is somewhat inevitable since the fonts **vgrind** uses are variable width.

The mechanism of **ctags**(1) in recognizing functions should be used here.

The **–w** option is a crock, but there is no other way to achieve the desired effect.

The macros defined in **tmac.vgrind** do not coexist gracefully with those of other macro packages, making filter mode difficult to use effectively.

NAME

vi, view – visual display editor based on ex(1)

SYNOPSIS

**vi** [ **–lrRx** ] [ **–t** *tag* ] [ **–w***nnn* ] [ *+command* ] *filename*...

**view**...

DESCRIPTION

**vi** (visual) is a display oriented text editor based on **ex**(1). **ex** and **vi** are, in fact, the same text editor; it is possible to get to the command mode of **ex** from within **vi** and vice-versa.

The **view** command runs **vi** with the **readonly** variable set. With **view**, you can browse through files interactively without making any changes.

OPTIONS

**–l** Set up for editing LISP programs.

**–r** Recover the named files after a crash.

**–R** Edit files in read only state. This has the same effect as the **view** command.

**–x** Prompt for a key with which to encrypt the file or files being edited.

**–t** *tag* Edit the file containing *tag*. There must be a tags database in the directory, built by **ctags**(1), that contains a reference to *tag*.

*+command*
Start the editing session by executing *command*.

*+command*
Start the editing session by executing *command*.

ENVIRONMENT

The editor recognizes the environment variable **EXINIT** as a command (or list of commands separated by | characters) to run when it starts up. If this variable is undefined, the editor checks for startup commands in the file **/.exrc** file, which you must own. However, if there is a **.exrc** owned by you in the current directory, the editor takes its startup commands from this file — overriding both the file in your home directory and the environment variable.

FILES

**ctags**(1), **ex**(1)

*Editing Text Files*
*Getting Started with SunOS: Beginner's Guide*

BUGS

Software TAB characters using CTRL-T work only immediately after the **autoindent**.

SHIFT-left and SHIFT-right on intelligent terminals do not make use of insert and delete character operations in the terminal.

The **wrapmargin** option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line will not be broken.

Repeating a change which wraps over the margin when **wrapmargin** is in effect does not generally work well: sometimes it just makes a mess of the change, and sometimes even leaves you in insert mode. A way to work around the problem is to replicate the changes using **y** (yank) and **p** (put).

Insert/delete within a line can be slow if TAB characters are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **':source'**; there is no way to use the **':append'**, **':change'**, and **':insert'** commands, since it is not possible to give more than one line of input to a **':'** escape. To use these on a **':global'** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with **vi** or **open**.

When using the **–r** option to recover a file, you must write the recovered text before quitting or you will lose it. **vi** does not prevent you from exiting without writing unless you make changes.

**vi** does not adjust when the SunView window in which it runs is resized.

RESTRICTIONS

The encryption facilities of **vi** are not available on software shipped outside the U.S.

**NAME**

vplot – plot graphics for a Versatec printer

**SYNOPSIS**

**vplot** [ **–VW** ] [ **–b** *lpr-argument* ] *filename*

**AVAILABILITY**

This command is available with the *Versatec Printer* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**vplot** reads **plot**(5) format graphics input from the file specified by *filename* (the standard input if no *filename* is specified) and produces a plot on the Varian or Versatec printer.

**OPTIONS**

**–V** Force output to the standard Versatec printer.

**–W** Force output to the (wide) Versatec printer rather than the standard Versatec printer.

**–b** *lpr-argument*
*argument* (the next argument on the command line) specifies extra arguments to **lpr**(1).

**SEE ALSO**

**lpr**(1), **plot**(1G), **plot**(5)

NAME

vswap – convert a foreign font file

SYNOPSIS

**/usr/lib/vswap [ –r ]**

AVAILABILITY

This command is available with the *Versatec Printer* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

Without the **-r** option, **vswap** translates its standard input (which must be a **vfont**(5) file in the reversed-byte order) into a locally correct *vfont* file on its standard output. With the **-r** option, **vswap** translates its standard input (which must be a **vfont** file in the local-byte order) into a byte-reversed **vfont** file on its standard output.

The UNIX system **vfont** representation for fonts is a binary file containing machine-dependent elements — short (16-bit) integers, in particular. There are (at least) two common ways of representing a 16-bit integer. A program compiled on a VAX will expect the VAX format, while the same program compiled on a machine using a Motorola 68000-family processor or a SPARC processor will expect the reverse format. **vswap** can be used to convert font files created on a VAX to the format required to use them on Suns. (All Suns use the same **vfont** format, regardless of the native byte order of the processor, including Suns that use the Intel 80386 processor. Programs that will be run on Suns that use the Intel 80386 processor must be aware of this, and convert the machine-dependent elements when they read or write **vfont** files.) It can also convert Sun-format font files to VAX format (with the **–r** option).

SEE ALSO

**troff**(1), **vfont**(5)

BUGS

A machine-independent font format should be defined.

**NAME**

vtroff – troff to a raster plotter

**SYNOPSIS**

**vtroff** [ **–wx** ] [ **–F** *majorfont* ] [ **–l***length* ] [ **–123** *minorfont* ] *troff-arguments*

**AVAILABILITY**

This command is available with the *Versatec Printer* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**vtroff** runs **troff**(1) sending its output through various programs to produce typeset output on a raster plotter such as a Benson-Varian or or a Versatec.

**OPTIONS**

**–w** Specify that a wide output device be used; the default is to use a narrow device.

**–x** Simulate photo-typesetter output exactly. As with, using the width tables for the C.A.T. photo-typesetter.

**–F** *fontname*

Specify *fontname* as the desired font. This will place normal, italic and bold versions of the font on positions 1, 2, and 3. The default font is a Hershey font. argument and then the font name.

**–l***length*

Split the output onto successive pages every *length* inches rather than the default 11 inches.

**–123** *minorfont*

Place a font only on a single position specified by *–n* (where *n* is **1, 2,** or **3**) and the minor font name. A **.r** will be added to the minor font name if needed. Thus

**vtroff –ms paper**

will set a paper in the Hershey font, while

**vtroff –F nonie –ms paper**

will set the paper in the (sans serif) nonie font.

**FILES**

| | |
|---|---|
| **/usr/share/lib/tmac/tmac.vcat** | default font mounts and bug fixes |
| **/usr/lib/fontinfo/*** | fixes for other fonts |
| **/usr/lib/vfont** | directory containing fonts |

**SEE ALSO**

**troff**(1), **vfont**(5)

**BUGS**

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Versatec fonts have different widths for individual characters than the fonts found on the typesetter, the following dodge was necessary: If you do not use the **.fp troff** directive then you get the widths of the standard typesetter fonts suitable for shipping the output of troff over the network to the computer center A machine for phototypesetting. If, however, you remount the R, I, B and S fonts, then you get the width tables for the Versatec.

NAME

vwidth – make a troff width table for a font

SYNOPSIS

**/usr/lib/vwidth** *fontfile pointsize* > **ftxx.c**

**cc -c ftxx.c**

**mv ftxx.o /usr/lib/font/ftxx**

AVAILABILITY

This command is available with the *Versatec Printer* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**vwidth** translates from the width information stored in the **vfont** style format to the format expected by **troff**(1) — an object file in **a.out**(5) format. **troff** should look directly in the font file but it doesn't.

**vwidth** should be used after editing a font with **fontedit**(1). It is not necessary to use **vwidth** unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use **vwidth** if the physical width of the glyph (for instance the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and **vwidth** run.

**vwidth** produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the **.o** file) saved. The resulting file should be placed in **/usr/lib/font** in the file ft*xx* where *xx* is a one or two letter code that is the logical (internal to **troff**) font name. This name can be found by looking in the file **/usr/lib/fontinfo**/*fname* * where *fname* is the external name of the font.

FILES

**/usr/lib/font**
**a.out**

SEE ALSO

**troff**(1), **vtroff**(1), **fontedit**(1), **a.out**(5), **vfont**(5)

NAME

w – who is logged in, and what are they doing

SYNOPSIS

**w [ –hs ] [ user ]**

DESCRIPTION

w displays a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields displayed are: the users login name, the name of the tty the user is on, the time of day the user logged on (in hours:minutes), the idle time — that is, the number of minutes since the user last typed anything (in hours:minutes), the CPU time used by all processes and their children on that terminal (in minutes:seconds), the CPU time used by the currently active processes (in minutes:seconds), the name and arguments of the current process.

If a **user** name is included, output is restricted to that user.

OPTIONS

**–h** Suppress the heading.

**–s** Produce a short form of output. In the short form, the tty is abbreviated, the login time and CPU times are left off, as are the arguments to commands.

**–l** Produce a long form of output, which is the default.

EXAMPLE

```
example% w
7:36am  up 6 days, 16:45,  1 users,  load average: 0.20, 0.23, 0.18
User     tty      login@  idle    JCPU   PCPU  what
ralph    console  7:10am     1      10:05  4:31  w
example%
```

FILES

**/etc/utmp**
**/dev/kmem**
**/dev/drum**

SEE ALSO

**ps**(1), **who**(1), **utmp**(5)

BUGS

The notion of the ''current process'' is muddy. The current algorithm is 'the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal'. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. In cases where no process can be found, w prints '–'.

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is ''charged'' with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

w does not know about the new conventions for detecting background jobs. It will sometimes find a background job instead of the right one.

**NAME**

wait – wait for a process to finish

**SYNOPSIS**

**wait**

**DESCRIPTION**

Wait until all processes started with **&** or **bg** have completed, and report on abnormal terminations.

Because the **wait**(2) system call must be executed in the parent process, the shell itself executes **wait**, without creating a new process.

**SEE ALSO**

**csh**(1), **sh**(1), **wait**(2)

**BUGS**

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for. This bug does not apply to **csh**(1).

NAME

wall – write to all users logged in

SYNOPSIS

**wall** [ **–a** ] [ *filename* ]

DESCRIPTION

**wall** reads the standard input until an EOF. It then sends this message, preceded by '**Broadcast Message ...**', to all logged in users. If *filename* is given, then the message is read in from that file. Normally, pseudo-terminals that do not correspond to rlogin sessions are ignored. Thus when in **sunview**(1), the message appears only on the console window. However, **–a** will send the message even to such pseudo-terminals.

The sender should be super-user to override any protections the users may have invoked.

FILES

**/dev/tty?**
**/etc/utmp**

SEE ALSO

**mesg**(1), **sunview**(1), **write**(1)

NAME

wc – display a count of lines, words and characters

SYNOPSIS

**wc** [ **–lwc** ] [ *filename* ... ]

DESCRIPTION

**wc** counts lines, words, and characters in *filename*s, or in the standard input if no *filename* appears. It also keeps a total count for all named files. A word is a string of characters delimited by SPACE, TAB, or NEW-LINE characters.

OPTIONS

When *filename*s are specified on the command line, their names will be printed along with the counts.

The default is **–lwc** (count lines, words, and characters).

**l** Count lines.

**w** Count words.

**c** Count characters.

EXAMPLE

```
example%
wc /usr/share/man/man1/{csh.1,sh.1,telnet.1}
    1876   11223   65895  /usr/share/man/man1/csh.1
     674    3310   20338  /usr/share/man/man1/sh.1
     260    1110    6834  /usr/share/man/man1/telnet.1
    2810   15643   93067  total
example%
```

## NAME

what – identify the version of files under SCCS

## SYNOPSIS

**what** *filename*

## DESCRIPTION

**what** searches the given *filename*s for all occurrences of the pattern that **get**(1) substitutes for **%Z%** (this is **@(#)** at this printing) and prints out what follows until the first , >, NEWLINE, \, or NULL character. For example, if the C program in file **program.c** contains

```
char ident[ ] = " @(#)identification information ";
```

and **program.c** is compiled to yield **program.o** and **a.out**, the command

```
what f.c f.o a.out
```

will print

```
f.c:    identification information
f.o:    identification information
a.out:  identification information
```

**what** is intended to be used in conjunction with the SCCS command **get**(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

## SEE ALSO

**file**(1), **get**(1), **help**(1), **sccs**(1)

*Programming Utilities and Libraries*

## DIAGNOSTICS

Use **help**(1) for explanations.

## BUGS

It is possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

whatis – display a one-line summary about a keyword

SYNOPSIS

**whatis** *command...*

DESCRIPTION

**whatis** looks up a given *command* and displays the header line from the manual section. You can then run the **man**(1) command to get more information. If the line starts '**name**(*section*) ...' you can do '**man** *section* **name**' to get the documentation for it. Try '**whatis ed**' and then you should do '**man 1 ed**' to get the manual page for **ed**(1).

**whatis** is actually just the **–f** option to the **man**(1) command.

FILES

**/usr/share/man/whatis** data base

SEE ALSO

**man**(1), **catman**(8)

## NAME

whereis – locate the binary, source, and manual page files for a command

## SYNOPSIS

**whereis** [ **–bmsu** ] [ **–BMS** *directory*... **–f** ] *filename* ...

## DESCRIPTION

**whereis** locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form *.ext*, for example, **.c**. Prefixes of **s.** resulting from use of source code control are also dealt with. **whereis** then attempts to locate the desired program in a list of standard places:

**/usr/bin**
**/usr/bin**
**/usr/5bin**
**/usr/games**
**/usr/hosts**
**/usr/include**
**/usr/local**
**/usr/etc**
**/usr/lib**
**/usr/share/man**
**/usr/src**
**/usr/ucb**

## OPTIONS

**–b** Search only for binaries.

**–m** Search only for manual sections.

**–s** Search only for sources.

**–u** Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus '**whereis –m –u ***' asks for those files in the current directory which have no documentation.

**–B** Change or otherwise limit the places where **whereis** searches for binaries.

**–M** Change or otherwise limit the places where **whereis** searches for manual sections.

**–S** Change or otherwise limit the places where **whereis** searches for sources.

**–f** Terminate the last directory list and signals the start of file names, and *must* be used when any of the **–B**, **–M**, or **–S** options are used.

## EXAMPLE

Find all files in **/usr/bin** which are not documented in **/usr/share/man/man1** with source in **/usr/src/cmd**:

```
example% cd /usr/ucb
example% whereis –u –M /usr/share/man/man1 –S /usr/src/cmd –f *
```

## FILES

**/usr/src/***
**/usr/{doc,man}/***
**/etc, /usr/{lib,bin,ucb,old,new,local}**

## SEE ALSO

**chdir**(2)

**BUGS**

Since **whereis** uses **chdir**(2) to run faster, pathnames given with the **–M**, **–S**, or **–B** must be full; that is, they must begin with a '/'.

**NAME**

which – locate a command; display its pathname or alias

**SYNOPSIS**

**which** [ *filename* ] ...

**DESCRIPTION**

**which** takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's **.cshrc** file.

**FILES**

| | |
|---|---|
| **/.cshrc** | source of aliases and path values |

**SEE ALSO**

**csh**(1)

**DIAGNOSTICS**

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

**BUGS**

Only aliases and paths from **/.cshrc** are used; importing from the current environment is not attempted. Must be executed by **csh**(1), since only **csh** knows about aliases.

To compensate for **/.cshrc** files in which aliases depend upon the **prompt** variable being set, **which** sets this variable. If the **/.cshrc** produces output or prompts for input when **prompt** is set, **which** may produce some strange results.

NAME

who – who is logged in on the system

SYNOPSIS

**who** [ **who-file** ] [ **am i** ]

DESCRIPTION

Used without arguments, **who** lists the login name, terminal name, and login time for each current user. **who** gets this information from the **/etc/utmp** file.

If a filename argument is given, the named file is examined instead of **/etc/utmp**. Typically the named file is **/var/adm/wtmp**, which contains a record of all logins since it was created. In this case, **who** lists logins, logouts, and crashes. Each login is listed with user name, terminal name (with **/dev/** suppressed), and date and time. Logouts produce a similar line without a user name. Reboots produce a line with ' ' in place of the device name, and a fossil time indicating when the system went down. Finally, the adjacent pair of entries '|' and '}' indicate the system-maintained time just before and after a **date** command changed the system's idea of the time.

With two arguments, as in '**who am i**' (and also '**who is who**'), **who** tells who you are logged in as: it displays your hostname, login name, terminal name, and login time.

EXAMPLES

```
example% who am i
example!ralph   ttyp0    Apr 27 11:24
example%

example% who
mktg      ttym0  Apr 27 11:11
joe       ttyp0  Apr 27 11:25
ralph     ttyp1  Apr 27 11:30
example%
```

FILES

**/etc/utmp**
**/var/adm/wtmp**

SEE ALSO

**w**(1), **whoami**(1), **utmp**(5)

NAME

whoami – display the effective current username

SYNOPSIS

**whoami**

DESCRIPTION

**whoami** displays your login name; **whoami** works even if you have used **su**(1) to temporarily adopt another user ID, since it gets its information from the **/etc/utmp** file.

FILES

| | |
|---|---|
| **/etc/passwd** | username data base |
| **/etc/utmp** | database of users currently logged in |

SEE ALSO

**su**(1), **who**(1), **utmp**(5)

NAME

whois – DARPA Internet user name directory service

SYNOPSIS

**whois** [ **–h** *host* ] *identifier*

DESCRIPTION

**whois** searches for an ARPANET directory entry for an *identifier* which is either a name (such as "Smith") or a handle (such as "SRI-NIC"). You can force a name-only search by preceding the name with a period; you can force a handle-only search by preceding the handle with an exclamation point. See EXAMPLES.

If you are searching for a group or organization entry, you can have the entire membership list of the group displayed with the record by preceding the argument with '*' ( an asterisk).

You may of course use an exclamation point and asterisk, or a period and asterisk together.

EXAMPLES

**example% whois Smith**

looks for name or handle SMITH.

**example% whois !SRI-NIC**

looks for handle SRI-NIC only.

**example% whois .Smith, John**

looks for name JOHN SMITH only.

Adding '...' to the name or handle argument will match anything from that point; that is, '**ZU...**' will match **ZUL, ZUM,** etc.

NAME

write – write a message to another user

SYNOPSIS

**write** *username* [ *ttyname* ]

DESCRIPTION

**write** copies lines from your standard input to *user*'s screen.

When you type a **write** command, the person you are writing to sees a message like this:

**Message from** *hostname*!*yourname* **on** *yourttyname*

After typing the **write** command, enter the text of your message. What you type appears line-by-line on the other user's screen. Conclude by typing an EOF indication (CTRL-D) or an interrupt. At this point **write** displays EOT on your recipient's screen and exits.

To write to a user who is logged in more than once, use the *ttyname* argument to indicate the appropriate terminal name.

You can grant or deny other users permission to write to you by using the **mesg** command (default allows writing). Certain commands, **nroff**(1) and **pr**(1V) in particular, do not allow anyone to write to you while you are using them in order to prevent messy output.

If **write** finds the character '**!**' at the beginning of a line, it calls the shell to execute the rest of the line as a command.

Two people can carry on a conversation by writing" to each other. When the other person receives the message indicating you are writing to him, he can then **write** back to you if he wishes. However, since you are now simultaneously typing and receiving messages, you end up with garbage on your screen unless you work out some sort of scheduling scheme with your partner. You might try the following conventional protocol: when you first write to another user, wait for him to write back before starting to send. Each person should end each message with a distinctive signal — **-o-** (for over") is standard — so that the other knows when to begin a reply. To end your conversation, type **-oo-** (for over and out") before finishing the conversation.

EXAMPLE

Here is an example of a short dialog between two people on different terminals. Two users called Horace" and Eudora" are logged in on a system called jones". To illustrate the process, both users' screens are shown side-by-side:

| Eudora's Terminal | Horace's Terminal |
|---|---|
| | *Horace is staring at his screen* |
| **jones% write horace** | |
| **how about a squash game tonight? -o-** | |
| | **Message from jones!eudora on tty09 at 17:05 ...** |
| | **how about a squash game tonight? -o-** |
| | **jones% write eudora** |
| | **I'm playing tiddlywinks with Carmeline -o-** |
| **Message from jones!horace on tty03 at 17:06 ...** | |
| **I'm playing tiddlywinks with Carmeline -o-** | |
| **How about the beach on Sunday? -o-** | |
| | **How about the beach on Sunday? -o-** |
| | **Sorry, I'm washing my tent that day -o-** |
| **Sorry, I'm washing my tent that day -o-** | |
| **See you when I get back from Peru -oo-** | |
| | **See you when I get back from Peru -oo-** |
| **^D** | |
| **jones%** | **EOF** |
| | **I hear rack of llama is very tasty -oo-** |
| | **^D** |

**I hear rack of llama is very tasty -oo-**
**EOF** **jones%**

FILES

| | |
|---|---|
| **/etc/utmp** | to find user |
| **/usr/bin/sh** | to execute ! |

SEE ALSO

**mail**(1), **mesg**(1), **pr**(1V), **talk**(1), **troff**(1), **who**(1)

## NAME

xargs – construct the arguments list(s) and execute a command

## SYNOPSIS

**xargs** [ **–ptx** ] [ **–l***number* ] [ **–i***replstr* ] [ **–n***number* ] [ **–s***size* ] [ **–e***eofstr* ]
[ *command* [ *initial-arguments* ] ]

## AVAILABILITY

This command is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**xargs** combines the fixed *initial-arguments* with arguments read from the standard input, to execute the specified *command* one or more times. The number of arguments read for each *command* invocation, and the manner in which they are combined are determined by the options specified.

*command*, which may be a shell file, is searched for using one's **$PATH**. If *command* is omitted, **/usr/bin/echo** is used.

Arguments read in from the standard input are defined to be contiguous strings of characters delimited by white space. Empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if they are escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings, a '\' (backslash) will escape the character it precedes.

Each arguments-list is constructed starting with the *initial-arguments*, followed by some number of arguments read from the standard input (Exception: see **–i** option). Options **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these options are coded, the *initial-arguments* are followed by arguments read continuously from the standard input until an internal buffer is full, and then *command* is executed with the accumulated arguments. This process is repeated until there are none left. When there are option conflicts (for instance, **–l** versus **–n**), the last option takes precedence.

**xargs** will terminate if it receives a return code of **–1**, or if it cannot execute *command*. When *command* is a shell script, it should explicitly **exit** (see **sh**(1)) with an appropriate value to avoid accidentally returning with **–1**.

## OPTIONS

**–p** Prompt mode. The user is asked whether to execute *command* each invocation. Trace mode (**–t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

**–t** Trace mode. The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

**–x** Terminate **xargs** if any argument list would be greater than **size** characters; **–x** is forced by the options **–i** and **–l**. When neither of the options **–i**, **–l**, or **–n** are coded, the total length of all arguments must be within the **size** limit.

**–l***number*

*command* is executed for each nonempty *number* lines of arguments from the standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first NEWLINE *unless* the last character of the line is a SPACE or a TAB; a trailing SPACE/TAB signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **–x** is forced.

**–i***replstr*

Insert mode: *command* is executed for each line from the standard input, taking the entire line as a single argument, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. SPACE and TAB characters at the beginning of each line are thrown away. Constructed arguments may not

grow larger than 255 characters, and option **–x** is also forced. **{}** is assumed for *replstr* if not specified.

**–n***number*
Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than **size** characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **–x** is also coded, each *number* arguments must fit in the **size** limitation, else **xargs** terminates execution.

**–s***size* The maximum total size of each argument list is set to **size** characters; **size** must be a positive integer less than or equal to 470. If **–s** is not coded, 470 is taken as the default. Note: the character count for **size** includes one extra character for each argument and the count of characters in the command name.

**–e***eofstr*
*eofstr* is taken as the logical EOF string. '_' (underbar) is assumed for the logical **EOF** string if **–e** is not coded. The value **–e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). **xargs** reads the standard input until either EOF or the logical **EOF** string is encountered.

## EXAMPLES

The following will move all files from directory **$1** to directory **$2**, and echo each move command just before doing it:

```
ls $1 | xargs –i –t
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file **log**:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into **arch**(1) 1.) one at a time, or 2.) many at a time.

```
1.  ls | xargs –p –l ar r arch
2.  ls | xargs –p –l | xargs ar r arch
```

The following will execute **diff**(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs –n2 diff
```

## FILES

**/usr/bin/echo**

## SEE ALSO

**arch**(1), **diff**(1), **sh**(1)

NAME

xsend, xget, enroll – send or receive secret mail

SYNOPSIS

**xsend** *username*

**xget**

**enroll**

DESCRIPTION

These commands implement a secure communication channel, which is like **mail**(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use **enroll**; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use **xget**. It asks for your password, then gives you the messages.

To send secret mail, use **xsend** in the same manner as the ordinary mail command. Unlike **mail**, **xsend** accepts only one target. A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

**/var/spool/secretmail/*.key** keys
**/var/spool/secretmail/*.[0-9]** messages

SEE ALSO

**mail**(1)

BUGS

The knapsack public-key cryptosystem is known to be breakable.

Secret mail should be integrated with ordinary mail.

The announcement of secret mail makes "traffic analysis" possible.

RESTRICTIONS

These facilities are not available on software shipped outside the U.S.

NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

**xstr** [ – ] [ **–cv** ] [ **–l** array ] [ *filename* ]

DESCRIPTION

**xstr** maintains a file called **strings** into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, which are most useful if they are also read-only.

The command

**xstr –c name**

extracts the strings from the C source in name, replacing string references by expressions of the form **&xstr**[*number*] for some number. An appropriate declaration of **xstr** is prepended to the file. The resulting C text is placed in the file **x.c**, to then be compiled. The strings from this file are placed in the **strings** data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file **xs.c** declaring the common **xstr** space can be created by a command of the form

**xstr**

This **xs.c** file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

**xstr** can also be used on a single file. A command

**xstr** name

creates files **x.c** and **xs.c** as before, without using or affecting any **strings** file in the same directory.

It may be useful to run **xstr** after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. **xstr** reads from the standard input when the argument '–' is given. An appropriate command sequence for running **xstr** after the C preprocessor is:

```
cc –E name.c | xstr –c –
cc –c x.c
mv x.o name.o
```

**xstr** does not touch the file **strings** unless new items are added; thus **make**(1) can avoid remaking **xs.o** unless truly necessary.

OPTIONS

–c*filename*
: Take C source text from *filename*.

–v Verbose: display a progress report indicating where new or duplicate strings were found.

–l *array*
: Specify the named *array* in program references to abstracted strings. The default array name is **xstr**.

FILES

| | |
|---|---|
| **strings** | data base of strings |
| **x.c** | massaged C source |
| **xs.c** | C source for definition of array ''xstr'' |
| **/tmp/xs*** | temp file when "xstr name" doesn't touch **strings** |

**SEE ALSO**

**make**(1), **mkstr**(1)

**BUGS**

If a string is a suffix of another string in the data base, but the shorter string is seen first by **xstr** both strings will be placed in the data base, when just placing the longer one there would do.

## NAME

yacc – yet another compiler-compiler: parsing program generator

## SYNOPSIS

**yacc** [ **–dv** ] *grammar*

## DESCRIPTION

**yacc** converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a function named **yyparse**( ). The **yyparse**( ) function must be loaded with the lexical analyzer **yylex**( ), as well as **main**( ) and **yyerror**( ), an error handling routine. These routines must be supplied by the user; **lex**(1) is useful for creating lexical analyzers usable by **yacc**-produced parsers.

## OPTIONS

**–d** Generate the file **y.tab.h** with the **define** statements that associate the **yacc**-assigned "token codes" with the user-declared "token names" so that source files other than **y.tab.c** can access the token codes.

**–v** Prepare the file **y.output** containing a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

## FILES

| | |
|---|---|
| **.y.output** | description of parsing tables and conflict report |
| **y.tab.c** | output parser |
| **y.tab.h** | defines for token names |
| **yacc.tmp, yacc.acts** | temporary files |
| **/usr/lib/yaccpar** | parser prototype for C programs |

## SEE ALSO

**lex**(1)

**yacc** in *Programming Utilities and Libraries*
*LR Parsing* by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974

## DIAGNOSTICS

The number of reduce-reduce and SHIFT-reduce conflicts is reported on the standard output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

## BUGS

Because file names are fixed, at most one **yacc** process should be active in a given directory at a time.

**NAME**

yes – be repetitively affirmative

**SYNOPSIS**

**yes** [ *expletive* ]

**DESCRIPTION**

**yes** repeatedly outputs **y**, or if *expletive* is given, that is output repeatedly. Termination is by typing an interrupt character.

NAME

ypcat - print values in a YP data base

SYNOPSIS

**ypcat** [ **–kt** ] [ **–d** *domainname* ] *mname*

**ypcat –x**

AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**ypcat** prints out values in a Yellow Pages (YP) map specified by *mname*, which may be either a map name or a map nickname . Since **ypcat** uses the YP network services, no YP server is specified.

To look at the network-wide password database, **passwd.byname**, (with the nickname **passwd**). type in:

**ypcat passwd**

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the Yellow Pages.

OPTIONS

**–k** Display the keys for those maps in which the values are null or the key is not part of the value. (None of the maps derived from files that have an ASCII version in **/etc** fall into this class.)

**–t** Inhibit translation of *mname* to map name. For example, '**ypcat –t passwd**' will fail because there is no map named **passwd**, whereas ' **ypcat passwd** ' will be translated to 'ypcat passwd.byname'.

**–d** *domainname*
Specify a domain other that the default domain. The default domain is returned by *domainname*.

**–x** Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

**domainname**(1), **ypmatch**(1), **ypfiles**(5), **ypserv**(8)

**NAME**

ypmatch - print the value of one or more keys from a YP map

**SYNOPSIS**

**ypmatch** [ **–d** *domain* ] [ **–k** ] [ **–t** ] *key* ... *mname*

**ypmatch –x**

**AVAILABILITY**

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**ypmatch** prints the values associated with one or more keys from the Yellow Pages (YP) map specified by *mname*, which may be either a *mapname* or an map nickname.

Multiple keys can be specified; the same map will be searched for all . The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

**OPTIONS**

**–d** Specify a domain other than the default domain.

**–k** Before printing the value of a key, print the key itself, followed by a ':' colon . This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

**–t** Inhibit translation of nickname to *mapname*. For example, '**ypmatch –t zippy passwd** ' will fail because there is no map named **passwd**, while '**ypmatch zippy passwd** ' will be translated to 'ypmatch zippy passwd.byname '.

**–x** Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

**SEE ALSO**

**ypcat**(1), **ypfiles**(5)

NAME

yppasswd – change your network password in the Yellow Pages

SYNOPSIS

**yppasswd** [ *name* ]

AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**yppasswd** changes (or installs) a network password associated with the user *name* (your own name by default) in the Yellow Pages. The Yellow Pages password may be different from the one on your own machine.

**yppasswd** prompts for the old Yellow Pages password, and then for the new one. You must type in the old password correctly for the change to take effect. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long, if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough. Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

The Yellow Pages password daemon, **yppasswdd**(8C) must be running on your YP server in order for the new password to take effect.

SEE ALSO

**passwd**(1), **ypfiles**(5), **yppasswdd**(8C)

BUGS

The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

NAME

ypwhich – which host is the YP server or map master?

SYNOPSIS

**ypwhich** [ **–d** [ *domain* ] ] [ **–V1** | **–V2** ] [ *hostname* ]
**ypwhich** [ **–t** *mapname* ] [ **–d** *domain* ] **–m** [ *mname* ]
**ypwhich –x**

AVAILABILITY

This command is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**ypwhich** tells which YP server supplies Yellow Pages services to a YP client, or which is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP master it is using.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the Yellow Pages.

OPTIONS

**–d** Use *domain* instead of the default domain.

**–V1** Which server is serving v.1 YP protocol-speaking client processes.

**–V2** Which server is serving v.2 YP protocol client processes.

If neither version is specified, **ypwhich** attempts attempts to locate the server that supplies the (current) v.2 services. If there is no v.2 server currently bound, **ypwhich** then attempts to locate the server supplying the v.1 services. Since YP servers and YP clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

**–t** *mapname*
Inhibit nickname translation; useful if there is a *mapname* identical to a nickname. This is not true of any Sun-supplied map.

**–m** Find the master YP server for a map. No *hostname* can be specified with **–m**. *mname* can be a mapname, or a nickname for a map. When *mname* is omitted, produce a list available maps.

**–x** Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

**ypfiles**(5), **rpcinfo**(8C), **ypserv**(8), **ypset**(8)

NAME

intro – introduction to system calls and error numbers

SYNOPSIS

**#include <errno.h>**

DESCRIPTION

This section describes all of the system calls. A '(2V)' heading indicates that the system call performs differently when called from programs that use the System V libraries (programs compiled using **/usr/5bin/cc**). On these pages, both the regular behavior and the System V behavior is described.

Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible return value. This is almost always '**−1**'; the individual descriptions specify the details. An error number is also made available in the external variable **errno**. **errno** is not cleared on successful calls, so it should be tested only after an error has been indicated. Note: a number of system calls overload the meanings of these error numbers, and the meanings must be interpreted according to the type and circumstances of the call.

As with normal arguments, all return codes and values from functions are of type integer unless otherwise noted.

Each system call description attempts to list all possible error numbers. The following is a complete list of the error numbers and their names as given in **<errno.h>**.

Error 0 Unused.

1 EPERM Not owner

Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or super-user. It is also returned for attempts by ordinary users to do things allowed only to the super-user.

2 ENOENT No such file or directory

This error occurs when a filename is specified and the file should exist but does not, or when one of the directories in a pathname does not exist.

3 ESRCH No such process

The process or process group whose number was given does not exist, or any such process is already dead.

4 EINTR Interrupted system call

An asynchronous signal (such as interrupt or quit) that the process has elected to catch occurred during a system call. If execution is resumed after processing the signal, and the system call is not restarted, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error

Some physical I/O error occurred. This error may in some cases occur on a call following the one to which it actually applies.

6 ENXIO No such device or address

I/O on a special file refers to a subdevice that does not exist, or beyond the limits of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.

7 E2BIG Arg list too long

An argument list longer than 1,048,576 bytes is presented to **execve**(2) or a routine that called **execve**( ).

8 ENOEXEC Exec format error

A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid magic number (see **a.out**(5)).

9 EBADF Bad file number

Either a file descriptor refers to no open file, or a read (respectively, write) request is made to a file that is open only for writing (respectively, reading).

10 ECHILD No children
A **wait**(2) was executed by a process that had no existing or unwaited-for child processes.

11 EAGAIN No more processes
A **fork**(2) failed because the system's process table is full or the user is not allowed to create any more processes, or a system call failed because of insufficient resources.

12 ENOMEM Not enough memory
During an **execve**(2), **sbrk**(), or **brk**(2), a program asks for more address space or swap space than the system is able to supply, or a process size limit would be exceeded. A lack of swap space is normally a temporary condition; however, a lack of address space is not a temporary condition. The maximum size of the text, data, and stack segments is a system parameter. Soft limits may be increased to their corresponding hard limits.

13 EACCES Permission denied
An attempt was made to access a file in a way forbidden by the protection system.

14 EFAULT Bad address
The system encountered a hardware fault in attempting to access the arguments of a system call.

15 ENOTBLK Block device required
A file that is not a block device was mentioned where a block device was required, for example, in **mount**(2).

16 EBUSY Device busy
An attempt was made to mount a file system that was already mounted or an attempt was made to dismount a file system on which there is an active file (open file, mapped file, current directory, or mounted-on directory).

17 EEXIST File exists
An existing file was mentioned in an inappropriate context, for example, **link**(2).

18 EXDEV Cross-device link
A hard link to a file on another file system was attempted.

19 ENODEV No such device
An attempt was made to apply an inappropriate system call to a device (for example, an attempt to read a write-only device) or an attempt was made to use a device not configured by the system.

20 ENOTDIR Not a directory
A non-directory was specified where a directory is required, for example, in a path prefix or as an argument to **chdir**(2).

21 EISDIR Is a directory
An attempt was made to write on a directory.

22 EINVAL Invalid argument
A system call was made with an invalid argument; for example, dismounting a non-mounted file system, mentioning an unknown signal in **sigvec**() or **kill**(), reading or writing a file for which **lseek**() has generated a negative pointer, or some other argument inappropriate for the call. Also set by math functions, see **intro**(3).

23 ENFILE File table overflow
The system's table of open files is full, and temporarily no more **open**() calls can be accepted.

24 EMFILE Too many open files
A process tried to have more open files than the system allows a process to have. The customary configuration limit is 64 per process.

25 ENOTTY Inappropriate ioctl for device
The code used in an **ioctl**() call is not supported by the object that the file descriptor in the call refers to.

26 *unused*

27 EFBIG File too large
The size of a file exceeded the maximum file size (1,082,201,088 bytes).

28 ENOSPC No space left on device"
A **write**() to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because no more disk blocks are available on the file system, or the allocation of an inode for a newly created file failed because no more inodes are available on the file system.

29 ESPIPE Illegal seek
An **lseek**() was issued to a socket or pipe. This error may also be issued for other non-seekable devices.

30 EROFS Read-only file system
An attempt to modify a file or directory was made on a file system mounted read-only.

31 EMLINK Too many links
An attempt was made to make more than 32767 hard links to a file.

32 EPIPE Broken pipe
An attempt was made to write on a pipe or socket for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is caught or ignored.

33 EDOM Math argument
The argument of a function in the math library (as described in section 3M) is out of the domain of the function.

34 ERANGE Result too large
The value of a function in the math library (as described in section 3M) is unrepresentable within machine precision.

35 EWOULDBLOCK Operation would block
An operation that would cause a process to block was attempted on an object in non-blocking mode (see **ioctl**(2)).

36 EINPROGRESS Operation now in progress
An operation that takes a long time to complete (such as a **connect**(2)) was attempted on a non-blocking object (see **ioctl**(2)).

37 EALREADY Operation already in progress
An operation was attempted on a non-blocking object that already had an operation in progress.

38 ENOTSOCK Socket operation on non-socket
Self-explanatory.

39 EDESTADDRREQ Destination address required
A required address was omitted from an operation on a socket.

40 EMSGSIZE Message too long
A message sent on a socket was larger than the internal message buffer.

41 EPROTOTYPE Protocol wrong type for socket
A protocol was specified that does not support the semantics of the socket type requested. For example, you cannot use the ARPA Internet UDP protocol with type SOCK_STREAM.

42 ENOPROTOOPT Option not supported by protocol
A bad option was specified in a **setsockopt**() or **getsockopt**(2) call.

43 EPROTONOSUPPORT Protocol not supported
The protocol has not been configured into the system or no implementation for it exists.

44 ESOCKTNOSUPPORT Socket type not supported
The support for the socket type has not been configured into the system or no implementation for it exists.

45 EOPNOTSUPP Operation not supported on socket
For example, trying to *accept* a connection on a datagram socket.

46 EPFNOSUPPORT Protocol family not supported
The protocol family has not been configured into the system or no implementation for it exists.

47 EAFNOSUPPORT Address family not supported by protocol family
An address incompatible with the requested protocol was used. For example, you should not necessarily expect to be able to use PUP Internet addresses with ARPA Internet protocols.

48 EADDRINUSE Address already in use
Only one usage of each address is normally permitted.

49 EADDRNOTAVAIL Can't assign requested address
Normally results from an attempt to create a socket with an address not on this machine.

50 ENETDOWN Network is down
A socket operation encountered a dead network.

51 ENETUNREACH Network is unreachable
A socket operation was attempted to an unreachable network.

52 ENETRESET Network dropped connection on reset
The host you were connected to crashed and rebooted.

53 ECONNABORTED Software caused connection abort
A connection abort was caused internal to your host machine.

54 ECONNRESET Connection reset by peer
A connection was forcibly closed by a peer. This normally results from the peer executing a **shutdown**(2) call.

55 ENOBUFS No buffer space available
An operation on a socket or pipe was not performed because the system lacked sufficient buffer space.

56 EISCONN Socket is already connected
A **connect**() request was made on an already connected socket; or, a **sendto**() or **sendmsg**() request on a connected socket specified a destination other than the connected party.

57 ENOTCONN Socket is not connected
An request to send or receive data was disallowed because the socket is not connected.

58 ESHUTDOWN Can't send after socket shutdown
A request to send data was disallowed because the socket had already been shut down with a previous **shutdown**(2) call.

59 *unused*

60 ETIMEDOUT Connection timed out
A *connect* request or an NFS request failed because the party to which the request was made did not properly respond after a period of time. (The timeout period is dependent on the communication protocol.)

61 ECONNREFUSED Connection refused
No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host.

62 ELOOP Too many levels of symbolic links
A pathname lookup involved more than 20 symbolic links.

63 ENAMETOOLONG File name too long
A component of a pathname exceeded 255 characters, or an entire pathname exceeded 1024 characters.

64 EHOSTDOWN Host is down
A socket operation failed because the destination host was down.

65 EHOSTUNREACH Host is unreachable
A socket operation was attempted to an unreachable host.

66 ENOTEMPTY Directory not empty
An attempt was made to remove a directory with entries other than '.' and '..' by performing a **rmdir()** system call or a **rename()** system call with that directory specified as the target directory.

67 *unused*

68 *unused*

69 EDQUOT Disc quota exceeded
A **write()** to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because the user's quota of disk blocks was exhausted, or the allocation of an inode for a newly created file failed because the user's quota of inodes was exhausted.

70 ESTALE Stale NFS file handle
An NFS client referenced a file that it had opened but that had since been deleted.

71 EREMOTE Too many levels of remote in path
An attempt was made to remotely mount a file system into a path that already has a remotely mounted component.

72 ENOSTR Not a stream device
A **putmsg**(2) or **getmsg**(2) system call was attempted on a file descriptor that is not a STREAMS device.

73 ETIME Timer expired
The timer set for a STREAMS **ioctl**(2) call has expired. The cause of this error is device specific and could indicate either a hardware or software failure, or perhaps a timeout value that is too short for the specific operation. The status of the **ioctl**(2) operation is indeterminate.

74 ENOSR Out of stream resources
During a STREAMS **open**(2V), either no STREAMS queues or no STREAMS head data structures were available.

75 ENOMSG No message of desired type
An attempt was made to receive a message of a type that does not exist on the specified message queue; see **msgop**(2).

76 EBADMSG Not a data message
During a **read**(2), **getmsg**(2), or **ioctl**(2) I_RECVFD system call to a STREAMS device, something has come to the head of the queue that cannot be processed. That something depends on the system call:

| | |
|---|---|
| **read**(2) | control information or a passed file descriptor. |
| **getmsg**(2) | passed file descriptor. |
| **ioctl**(2) | control or data information. |

77 EIDRM Identifier removed
This error is returned to processes that resume execution due to the removal of an identifier from the IPC system's name space (see **msgctl**(2), **semctl**(2), and **shmctl**(2)).

## DEFINITIONS

**Descriptor**

An integer assigned by the system when a file is referenced by **open**(2V), **dup**(2), or **pipe**(2) or a socket is referenced by **socket**(2) or **socketpair**(2) that uniquely identifies an access path to that file or socket from a given process or any of its children.

**Directory**

A directory is a special type of file that contains entries that are references to other files. Directory entries are called links. By convention, a directory contains at least two links, '.' and '..', referred to as *dot* and *dot-dot* respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.

**Effective User ID, Effective Group ID, and Access Groups**

Access to system resources is governed by three values: the effective user ID, the effective group ID, and the group access list.

The effective user ID and effective group ID are initially the process's real user ID and real group ID respectively. Either may be modified through execution of a set-user-ID or set-group-ID file (possibly by one of its ancestors) (see **execve**(2)).

The group access list is an additional set of group ID's used only in determining resource accessibility. Access checks are performed as described below in **File Access Permissions'**.

**File Access Permissions**

Every file in the file system has a set of access permissions. These permissions are used in determining whether a process may perform a requested operation on the file (such as opening a file for writing). Access permissions are established at the time a file is created. They may be changed at some later time through the **chmod**(2) call.

File access is broken down according to whether a file may be: read, written, or executed. Directory files use the execute permission to control if the directory may be searched.

File access permissions are interpreted by the system as they apply to three different classes of users: the owner of the file, those users in the file's group, anyone else. Every file has an independent set of access permissions for each of these classes. When an access check is made, the system decides if permission should be granted by checking the access information applicable to the caller.

Read, write, and execute/search permissions on a file are granted to a process if:

The process's effective user ID is that of the super-user.

The process's effective user ID matches the user ID of the owner of the file and the owner permissions allow the access.

The process's effective user ID does not match the user ID of the owner of the file, and either the process's effective group ID matches the group ID of the file, or the group ID of the file is in the process's group access list, and the group permissions allow the access.

Neither the effective user ID nor effective group ID and group access list of the process match the corresponding user ID and group ID of the file, but the permissions for "other users" allow access.

Otherwise, permission is denied.

**File Name**

Names consisting of up to 255 characters may be used to name an ordinary file, special file, or directory.

These characters may be selected from the set of all ASCII character excluding \0 (null) and the ASCII code for / (slash). (The parity bit, bit 8, must be 0.)

Note: it is generally unwise to use *, ?, [, or ] as part of filenames because of the special meaning attached to these characters by the shell. See **sh**(1). Although permitted, it is advisable to avoid the use of unprintable characters in filenames.

**Message Queue Identifier**

A message queue identifier (msqid) is a unique positive integer created by a **msgget**(2) system call. Each msqid has a message queue and a data structure associated with it. The data structure is referred to as **msqid_ds()** and contains the following members:

```
struct  ipc_perm msg_perm;  /* operation permission struct */
ushort  msg_qnum;           /* number of msgs on q */
ushort  msg_qbytes;         /* max number of bytes on q */
ushort  msg_lspid;          /* pid of last msgsnd operation */
ushort  msg_lrpid;          /* pid of last msgrcv operation */
time_t  msg_stime;          /* last msgsnd time */
time_t  msg_rtime;          /* last msgrcv time */
time_t  msg_ctime;          /* last change time */
                            /* Times measured in secs since */
                            /* 00:00:00 GMT, Jan. 1, 1970 */
```

**msg_perm()** is an ipc_perm structure that specifies the message operation permission (see below). This structure includes the following members:

```
ushort  cuid;               /* creator user id */
ushort  cgid;               /* creator group id */
ushort  uid;                /* user id */
ushort  gid;                /* group id */
ushort  mode;               /* r/w permission */
```

**msg_qnum** is the number of messages currently on the queue. **msg_qbytes** is the maximum number of bytes allowed on the queue. **msg_lspid** is the process ID of the last process that performed a **msgsnd** operation. **msg_lrpid** is the process ID of the last process that performed a **msgrcv** operation. **msg_stime** is the time of the last *msgsnd* operation, **msg_rtime** is the time of the last *msgrcv* operation, and **msg_ctime** is the time of the last **msgctl**(2) operation that changed a member of the above structure.

**Message Operation Permissions**

In the **msgop**(2) and **msgctl**(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

| | |
|---|---|
| 00400 | Read by user |
| 00200 | Write by user |
| 00060 | Read, Write by group |
| 00006 | Read, Write by others |

Read and Write permissions on a msqid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **msg_perm.[c]uid** in the data structure associated with *msqid* and the appropriate bit of the "user" portion (0600) of **msg_perm.mode** is set.

The effective user ID of the process does not match **msg_perm.[c]uid** and the effective group ID of the process matches **msg_perm.[c]gid** and the appropriate bit of the "group" portion (060) of **msg_perm.mode** is set.

The effective user ID of the process does not match **msg_perm.[c]uid** and the effective group ID of the process does not match **msg_perm.[c]gid** and the appropriate bit of the "other" portion (06) of **msg_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

**Parent Process ID**

A new process is created by a currently active process (see **fork**(2)). The parent process ID of a process is the process ID of its creator.

**Path Name and Path Prefix**

A pathname is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a filename. The total length of a pathname must be less than {MAXPATHLEN} (1024) characters.

More precisely, a pathname is a null-terminated character string constructed as follows:

*<path-name>::=<file-name>| <path-prefix><file-name>| /*
*<path-prefix>::=<rtprefix>| /<rtprefix>*
*<rtprefix>::=<dirname>/| <rtprefix><dirname>/*

where *<file-name>* is a string of 1 to 255 characters other than the ASCII slash and null, and *<dirname>* is a string of 1 to 255 characters (other than the ASCII slash and null) that names a directory.

If a pathname begins with a slash, the search begins at the *root* directory. Otherwise, the search begins at the current working directory.

A slash, by itself, names the root directory. A dot (.) names the current working directory.

A null pathname also refers to the current directory. However, this is not true of all UNIX systems. (On such systems, accidental use of a null pathname in routines that do not check for it may corrupt the current working directory.) For portable code, specify the current directory explicitly using '".."', rather than '""'.

**Process Group ID**

Each active process is a member of a process group that is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes (see **killpg**(2)) and the job control mechanisms of **csh**(1).

**Process ID**

Each active process in the system is uniquely identified by a positive integer called a process ID. The range of this ID is from 0 to 30000.

**Real User ID and Real Group ID**

Each user on the system is identified by a positive integer termed the real user ID.

Each user is also a member of one or more groups. One of these groups is distinguished from others and used in implementing accounting facilities. The positive integer corresponding to this distinguished group is termed the real group ID.

All processes have a real user ID and real group ID. These are initialized from the equivalent attributes of the process that created it.

**Root Directory and Current Working Directory**

Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. A process's root directory need not be the root directory of the root file system.

**Semaphore Identifier**

A semaphore identifier (semid) is a unique positive integer created by a **semget**(2) system call. Each semid has a set of semaphores and a data structure associated with it. The data structure is referred to as **semid_ds** and contains the following members:

```
struct   ipc_perm sem_perm;  /* operation permission struct */
ushort   sem_nsems;          /* number of sems in set */
time_t   sem_otime;          /* last operation time */
time_t   sem_ctime;          /* last change time */
                             /* Times measured in secs since */
                             /* 00:00:00 GMT, Jan. 1, 1970 */
```

**sem_perm** is an **ipc_perm** structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```
ushort   cuid;          /* creator user id */
ushort   cgid;          /* creator group id */
ushort   uid;           /* user id */
ushort   gid;           /* group id */
ushort   mode;          /* r/a permission */
```

The value of **sem_nsems** is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a **sem_num**. **sem_num** values run sequentially from 0 to the value of **sem_nsems** minus 1. **sem_otime** is the time of the last **semop**(2) operation, and **sem_ctime** is the time of the last **semctl**(2) operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort   semval;        /* semaphore value */
short    sempid;        /* pid of last operation */
ushort   semncnt;       /* # awaiting semval > cval */
ushort   semzcnt;       /* # awaiting semval = 0 */
```

**semval** is a non-negative integer. **sempid** is equal to the process ID of the last process that performed a semaphore operation on this semaphore. **semncnt** is a count of the number of processes that are currently suspended awaiting this semaphore's semval to become greater than its current value. **semzcnt** is a count of the number of processes that are currently suspended awaiting this semaphore's semval to become zero.

**Semaphore Operation Permissions**

In the **semop**(2) and **semctl**(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

```
00400          Read by user
00200          Alter by user
00060          Read, Alter by group
00006          Read, Alter by others
```

Read and Alter permissions on a semid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **sem_perm.[c]uid** in the data structure associated with *semid* and the appropriate bit of the "user" portion (0600) of **sem_perm.mode** is set.

The effective user ID of the process does not match **sem_perm.[c]uid** and the effective group ID of the process matches **sem_perm.[c]gid** and the appropriate bit of the "group" portion (060) of **sem_perm.mode** is set.

The effective user ID of the process does not match **sem_perm.[c]uid** and the effective group ID of the process does not match **sem_perm.[c]gid** and the appropriate bit of the "other" portion (06) of **sem_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

**Shared Memory Identifier**

A shared memory identifier (shmid) is a unique positive integer created by a **shmget**(2) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as **shmid_ds** and contains the following members:

```
struct  ipc_perm shm_perm;  /* operation permission struct */
int     shm_segsz;          /* size of segment */
ushort  shm_cpid;           /* creator pid */
ushort  shm_lpid;           /* pid of last operation */
short   shm_nattch;         /* number of current attaches */
time_t  shm_atime;          /* last attach time */
time_t  shm_dtime;          /* last detach time */
time_t  shm_ctime;          /* last change time */
                            /* Times measured in secs since */
                            /* 00:00:00 GMT, Jan. 1, 1970 */
```

**shm_perm** is an **ipc_perm** structure that specifies the shared memory operation permission (see below). This structure includes the following members:

```
ushort  cuid;               /* creator user id */
ushort  cgid;               /* creator group id */
ushort  uid;                /* user id */
ushort  gid;                /* group id */
ushort  mode;               /* r/w permission */
```

**shm_segsz** specifies the size of the shared memory segment. **shm_cpid** is the process ID of the process that created the shared memory identifier. **shm_lpid** is the process ID of the last process that performed a **shmop**(2) operation. **shm_nattch** is the number of processes that currently have this segment attached. **shm_atime** is the time of the last *shmat* operation, **shm_dtime** is the time of the last *shmdt* operation, and **shm_ctime** is the time of the last **shmctl**(2) operation that changed one of the members of the above structure.

**Shared Memory Operation Permissions**

In the **shmop**(2) and **shmctl**(2) system call descriptions, the permission required for an operation is given as "{token}", where "token" is the type of permission needed interpreted as follows:

| | |
|---|---|
| 00400 | Read by user |
| 00200 | Write by user |
| 00060 | Read, Write by group |
| 00006 | Read, Write by others |

Read and Write permissions on a shmid are granted to a process if one or more of the following are true:

The effective user ID of the process is super-user.

The effective user ID of the process matches **shm_perm.[c]uid** in the data structure associated with *shmid* and the appropriate bit of the "user" portion (0600) of **shm_perm.mode** is set.

The effective user ID of the process does not match **shm_perm.[c]uid** and the effective group ID of the process matches **shm_perm.[c]gid** and the appropriate bit of the "group" portion (060) of **shm_perm.mode** is set.

The effective user ID of the process does not match **shm_perm.[c]uid** and the effective group ID of the process does not match **shm_perm.[c]gid** and the appropriate bit of the "other" portion (06) of **shm_perm.mode** is set.

Otherwise, the corresponding permissions are denied.

**Sockets and Address Families**

A socket is an endpoint for communication between processes. Each socket has queues for sending and receiving data.

Sockets are typed according to their communications properties. These properties include whether messages sent and received at a socket require the name of the partner, whether communication is reliable, the format used in naming message recipients, etc.

Each instance of the system supports some collection of socket types; consult **socket**(2) for more information about the types available and their properties.

Each instance of the system supports some number of sets of communications protocols. Each protocol set supports addresses of a certain format. An Address Family is the set of addresses for a specific group of protocols. Each socket has an address chosen from the address family in which the socket was created.

**Special Processes**

The processes with a process ID's of 0, 1, and 2 are special. Process 0 is the scheduler. Process 1 is the initialization process **init**, and is the ancestor of every other process in the system. It is used to control the process structure. Process 2 is the paging daemon.

**Super-user**

A process is recognized as a *super-user* process and is granted special privileges if its effective user ID is 0.

**Tty Group ID**

Each active process can be a member of a terminal group that is identified by a positive integer called the tty group ID. This grouping is used to arbitrate between multiple jobs contending for the same terminal (see **csh**(1), and **termio**(4), and to terminate a group of related processes upon termination of one of the processes in the group (see **exit**(2) and **sigvec**(2)).

**STREAMS**

A set of kernel mechanisms that support the development of network services and data communication *driver*s. It defines interface standards for character input/output within the kernel and between the kernel and user level processes. The STREAMS mechanism is composed of utility routines, kernel facilities and a set of data structures.

**Stream**

A stream is a full-duplex data path within the kernel between a user process and driver routines. The primary components are a stream head, a *driver* and zero or more *modules* between the stream head and *driver*. A stream is analogous to a Shell pipeline except that data flow and processing are bidirectional.

**Stream Head**

In a stream, the stream head is the end of the stream that provides the interface between the stream and a user process. The principle functions of the stream head are processing STREAMS-related system calls, and passing data and information between a user process and the stream.

**Driver**

In a stream, the *driver* provides the interface between peripheral hardware and the stream. A *driver* can also be a pseudo-*driver*, such as a *multiplexor* or *emulator*, and need not be associated with a hardware device.

**Module**

A module is an entity containing processing routines for input and output data. It always exists in the middle of a stream, between the stream's head and a *driver*. A *module* is the STREAMS counterpart to the commands in a Shell pipeline except that a module contains a pair of functions which allow independent bidirectional (*downstream* and *upstream*) data flow and processing.

**Downstream**

In a stream, the direction from stream head to *driver*.

**Upstream**

In a stream, the direction from *driver* to stream head.

**Message**

In a stream, one or more blocks of data or information, with associated STREAMS control structures. Messages can be of several defined types, which identify the message contents. Messages are the only means of

transferring data and communicating within a stream.

**Message Queue**
In a stream, a linked list of *messages* awaiting processing by a *module* or *driver*.

**Read Queue**
In a stream, the *message queue* in a *module* or *driver* containing *messages* moving *upstream*.

**Write Queue**
In a stream, the *message queue* in a *module* or *driver* containing *messages* moving *downstream*.

**Multiplexor**
A multiplexor is a driver that allows STREAMS associated with several user processes to be connected to a single *driver*, or several *drivers* to be connected to a single user process. STREAMS does not provide a general multiplexing *driver*, but does provide the facilities for constructing them, and for connecting multiplexed configurations of STREAMS.

SEE ALSO

**csh**(1), **sh**(1), **brk**(2), **chdir**(2), **chmod**(2), **connect**(2), **dup**(2), **execve**(2), **exit**(2), **fork**(2), **getmsg**(2), **getsockopt**(2), **ioctl**(2), **killpg**(2), **link**(2), **mount**(2), **msgctl**(2), **msgget**(2), **msgop**(2), **open**(2V), **pipe**(2), **putmsg**(2), **read**(2), **semctl**(2), **semget**(2), **semop**(2), **setsockopt**(2), **shmctl**(2), **shmget**(2), **shmop**(2), **shutdown**(2), **sigvec**(2), **socket**(2), **socketpair**(2), **wait**(2), **intro(3)**, **perror(3)** **termio**(4), **a.out**(5)

LIST OF SYSTEM CALLS

| Name | Appears on Page | Description |
|---|---|---|
| **_exit()** | **exit**(2) | terminate a process |
| **accept()** | **accept**(2) | accept a connection on a socket |
| **access()** | **access**(2) | determine accessibility of file |
| **acct()** | **acct**(2) | turn accounting on or off |
| **adjtime()** | **adjtime**(2) | correct the time to allow synchronization of the system clock |
| **async_daemon()** | **nfssvc**(2) | NFS daemons |
| **audit()** | **audit**(2) | write a record to the audit log |
| **auditon()** | **auditon**(2) | manipulate auditing |
| **auditsvc()** | **auditsvc**(2) | write audit records to specified file descriptor |
| **bind()** | **bind**(2) | bind a name to a socket |
| **brk()** | **brk**(2) | change data segment size |
| **chdir()** | **chdir**(2) | change current working directory |
| **chmod()** | **chmod**(2) | change mode of file |
| **chown()** | **chown**(2) | change owner and group of a file |
| **chroot()** | **chroot**(2) | change root directory |
| **close()** | **close**(2) | delete a descriptor |
| **connect()** | **connect**(2) | initiate a connection on a socket |
| **creat()** | **creat**(2) | create a new file |
| **dup2()** | **dup**(2) | duplicate a descriptor |
| **dup()** | **dup**(2) | duplicate a descriptor |
| **execve()** | **execve**(2) | execute a file |
| **fchmod()** | **chmod**(2) | change mode of file |
| **fchown()** | **chown**(2) | change owner and group of a file |
| **fcntl()** | **fcntl**(2V) | file control |
| **flock()** | **flock**(2) | apply or remove an advisory lock on an open file |
| **fork()** | **fork**(2) | create a new process |
| **fstat()** | **stat**(2) | get file status |
| **fsync()** | **fsync**(2) | synchronize a file's in-core state with that on disk |
| **ftruncate()** | **truncate**(2) | set a file to a specified length |
| **getauid()** | **getauid**(2) | get and set user audit identity |
| **getdents()** | **getdents**(2) | gets directory entries in a filesystem independent format |
| **getdirentries()** | **getdirentries**(2) | gets directory entries in a filesystem independent format |
| **getdomainname()** | **getdomainname**(2) | get/set name of current domain |
| **getdtablesize()** | **getdtablesize**(2) | get descriptor table size |
| **getegid()** | **getgid**(2) | get group identity |
| **geteuid()** | **getuid**(2) | get user identity |
| **getgid()** | **getgid**(2) | get group identity |
| **getgroups()** | **getgroups**(2) | get or set group access list |
| **gethostid()** | **gethostid**(2) | get unique identifier of current host |
| **gethostname()** | **gethostname**(2) | get/set name of current host |
| **getitimer()** | **getitimer**(2) | get/set value of interval timer |
| **getmsg()** | **getmsg**(2) | get next message off a stream |
| **getpagesize()** | **getpagesize**(2) | get system page size |
| **getpeername()** | **getpeername**(2) | get name of connected peer |
| **getpgrp()** | **setpgrp**(2V) | set and/or return the process group of a process |
| **getpid()** | **getpid**(2) | get process identification |
| **getppid()** | **getpid**(2) | get process identification |
| **getpriority()** | **getpriority**(2) | get/set program scheduling priority |
| **getrlimit()** | **getrlimit**(2) | control maximum system resource consumption |
| **getrusage()** | **getrusage**(2) | get information about resource utilization |

| | | |
|---|---|---|
| **getsockname()** | **getsockname**(2) | get socket name |
| **getsockopt()** | **getsockopt**(2) | get and set options on sockets |
| **gettimeofday()** | **gettimeofday**(2) | get or set the date and time |
| **getuid()** | **getuid**(2) | get user identity |
| **ioctl()** | **ioctl**(2) | control device |
| **kill()** | **kill**(2V) | send a signal to a process or a group of processes |
| **killpg()** | **killpg**(2) | send signal to a process group |
| **link()** | **link**(2) | make a hard link to a file |
| **listen()** | **listen**(2) | listen for connections on a socket |
| **lseek()** | **lseek**(2) | move read/write pointer |
| **lstat()** | **stat**(2) | get file status |
| **mincore()** | **mincore**(2) | determine residency of memory pages |
| **mkdir()** | **mkdir**(2) | make a directory file |
| **mknod()** | **mknod**(2) | make a special file |
| **mmap()** | **mmap**(2) | map pages of memory |
| **mount()** | **mount**(2) | mount file system |
| **mprotect()** | **mprotect**(2) | set protection of memory mapping |
| **msgctl()** | **msgctl**(2) | message control operations |
| **msgget()** | **msgget**(2) | get message queue |
| **msgop()** | **msgop**(2) | message operations |
| **msgrcv()** | **msgop**(2) | message operations |
| **msgsnd()** | **msgop**(2) | message operations |
| **msync()** | **msync**(2) | synchronize memory with physical storage |
| **munmap()** | **munmap**(2) | unmap pages of memory. |
| **nfssvc()** | **nfssvc**(2) | NFS daemons |
| **open()** | **open**(2V) | open or create a file for reading or writing |
| **pipe()** | **pipe**(2) | create an interprocess communication channel |
| **poll()** | **poll**(2) | STREAMS input/output multiplexing |
| **profil()** | **profil**(2) | execution time profile |
| **ptrace()** | **ptrace**(2) | process trace |
| **putmsg()** | **putmsg**(2) | send a message on a stream |
| **quotactl()** | **quotactl**(2) | manipulate disk quotas |
| **read()** | **read**(2V) | read input |
| **readlink()** | **readlink**(2) | read value of a symbolic link |
| **readv()** | **read**(2V) | read input |
| **reboot()** | **reboot**(2) | reboot system or halt processor |
| **recv()** | **recv**(2) | receive a message from a socket |
| **recvfrom()** | **recv**(2) | receive a message from a socket |
| **recvmsg()** | **recv**(2) | receive a message from a socket |
| **rename()** | **rename**(2) | change the name of a file |
| **rmdir()** | **rmdir**(2) | remove a directory file |
| **sbrk()** | **brk**(2) | change data segment size |
| **select()** | **select**(2) | synchronous I/O multiplexing |
| **semctl()** | **semctl**(2) | semaphore control operations |
| **semget()** | **semget**(2) | get set of semaphores |
| **semop()** | **semop**(2) | semaphore operations |
| **send()** | **send**(2) | send a message from a socket |
| **sendmsg()** | **send**(2) | send a message from a socket |
| **sendto()** | **send**(2) | send a message from a socket |
| **setaudit()** | **setuseraudit**(2) | set the audit classes for a specified user ID |
| **setauid()** | **getauid**(2) | get and set user audit identity |
| **setdomainname()** | **getdomainname**(2) | get/set name of current domain |
| **setgroups()** | **getgroups**(2) | get or set group access list |

| | | |
|---|---|---|
| **sethostname()** | **gethostname**(2) | get/set name of current host |
| **setitimer()** | **getitimer**(2) | get/set value of interval timer |
| **setpgrp()** | **setpgrp**(2V) | set and/or return the process group of a process |
| **setpriority()** | **getpriority**(2) | get/set program scheduling priority |
| **setregid()** | **setregid**(2) | set real and effective group IDs |
| **setreuid()** | **setreuid**(2) | set real and effective user IDs |
| **setrlimit()** | **getrlimit**(2) | control maximum system resource consumption |
| **setsockopt()** | **getsockopt**(2) | get and set options on sockets |
| **settimeofday()** | **gettimeofday**(2) | get or set the date and time |
| **setuseraudit()** | **setuseraudit**(2) | set the audit classes for a specified user ID |
| **shmat()** | **shmop**(2) | shared memory operations |
| **shmctl()** | **shmctl**(2) | shared memory control operations |
| **shmdt()** | **shmop**(2) | shared memory operations |
| **shmget()** | **shmget**(2) | get shared memory segment identifier |
| **shmop()** | **shmop**(2) | shared memory operations |
| **shutdown()** | **shutdown**(2) | shut down part of a full-duplex connection |
| **sigblock()** | **sigblock**(2) | block signals |
| **sigpause()** | **sigpause**(2) | atomically release blocked signals and wait for interrupt |
| **sigsetmask()** | **sigsetmask**(2) | set current signal mask |
| **sigstack()** | **sigstack**(2) | set and/or get signal stack context |
| **sigvec()** | **sigvec**(2) | software signal facilities |
| **socket()** | **socket**(2) | create an endpoint for communication |
| **socketpair()** | **socketpair**(2) | create a pair of connected sockets |
| **stat()** | **stat**(2) | get file status |
| **statfs()** | **statfs**(2) | get file system statistics |
| **swapon()** | **swapon**(2) | add a swap device for interleaved paging/swapping |
| **symlink()** | **symlink**(2) | make symbolic link to a file |
| **sync()** | **sync**(2) | update super-block |
| **syscall()** | **syscall**(2) | indirect system call |
| **tell()** | **lseek**(2) | move read/write pointer |
| **truncate()** | **truncate**(2) | set a file to a specified length |
| **umask()** | **umask**(2) | set file creation mode mask |
| **uname()** | **uname**(2V) | get name of current system |
| **unlink()** | **unlink**(2) | remove directory entry |
| **unmount()** | **unmount**(2) | remove a file system |
| **utimes()** | **utimes**(2) | set file times |
| **vadvise()** | **vadvise**(2) | give advice to paging system |
| **vfork()** | **vfork**(2) | spawn new process in a virtual memory efficient way |
| **vhangup()** | **vhangup**(2) | virtually "hangup" the current control terminal |
| **wait3()** | **wait**(2) | wait for process to terminate or stop |
| **wait4()** | **wait**(2) | wait for process to terminate or stop |
| **wait()** | **wait**(2) | wait for process to terminate or stop |
| **WIFEXITED()** | **wait**(2) | wait for process to terminate or stop |
| **WIFSIGNALED()** | **wait**(2) | wait for process to terminate or stop |
| **WIFSTOPPED()** | **wait**(2) | wait for process to terminate or stop |
| **write()** | **write**(2V) | write output |
| **writev()** | **write**(2V) | write output |

## NAME

accept – accept a connection on a socket

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

ns = accept(s, addr, addrlen)
int ns, s;
struct sockaddr *addr;
int *addrlen;
```

## DESCRIPTION

The argument *s* is a socket that has been created with **socket**(2), bound to an address with **bind**(2), and is listening for connections after a **listen**(2). **accept**( ) extracts the first connection on the queue of pending connections, creates a new socket with the same properties of *s* and allocates a new file descriptor, *ns*, for the socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, **accept**( ) blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, **accept**( ) returns an error as described below. The accepted socket, *ns*, is used to read and write data to and from the socket which connected to this one; it is not used to accept more connections. The original socket *s* remains open for accepting further connections.

The argument *addr* is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the *addr* parameter is determined by the domain in which the communication is occurring. The *addrlen* is a value-result parameter; it should initially contain the amount of space pointed to by *addr*; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with SOCK_STREAM.

It is possible to **select**(2) a socket for the purposes of doing an **accept**( ) by selecting it for read.

## RETURN VALUE

The call returns −1 on error. If it succeeds, it returns a non-negative integer that is a descriptor for the accepted socket.

## ERRORS

The **accept**( ) will fail if:

| | |
|---|---|
| EBADF | The descriptor is invalid. |
| ENOTSOCK | The descriptor references a file, not a socket. |
| EOPNOTSUPP | The referenced socket is not of type SOCK_STREAM. |
| EFAULT | The *addr* parameter is not in a writable part of the user address space. |
| EWOULDBLOCK | The socket is marked non-blocking and no connections are present to be accepted. |

## SEE ALSO

**bind**(2), **connect**(2), **listen**(2), **select**(2), **socket**(2)

## NAME

access – determine accessibility of file

## SYNOPSIS

```
#include <sys/file.h>

#define R_OK    4    /* test for read permission */
#define W_OK    2    /* test for write permission */
#define X_OK    1    /* test for execute (search) permission */
#define F_OK    0    /* test for presence of file */

accessible = access(path, mode)
int accessible;
char *path;
int mode;
```

## DESCRIPTION

*path* points to a path name naming a file. **access( )** checks the named file for accessibility according to *mode*, which is an inclusive or of the bits **R_OK**, **W_OK** and **X_OK**. Specifying *mode* as **F_OK** (that is, 0) tests whether the directories leading to the file can be searched and the file exists.

The real user ID and the group access list (including the real group ID) are used in verifying permission, so this call is useful to set-UID programs.

The owner of a file has permission checked with respect to the *owner* read, write, and execute mode bits, members of the file's group other than the owner have permission checked with respect to the **group** mode bits, and all others have permissions checked with respect to the *other* mode bits.

Notice that only access bits are checked. A directory may be indicated as writable by **access**, but an attempt to open it for writing will fail (although files may be created there); a file may look executable, but **execve( )** will fail unless it is in proper format.

## RETURN VALUE

If *path* cannot be found or if any of the desired access modes would not be granted, then a −1 value is returned; otherwise a 0 value is returned.

## ERRORS

**access( )** to the file is denied if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The file named by *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EROFS | The file named by *path* is on a read-only file system and write access was requested. |
| EACCES | Permission bits of the file mode do not permit the requested access to the file named by *path*. |
| EFAULT | *path* points outside the process's allocated address space. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

## SEE ALSO

**chmod**(2), **stat**(2)

## NAME

acct – turn accounting on or off

## SYNOPSIS

**int acct (path)**
**char *path;**

## DESCRIPTION

**acct()** is used to enable or disable the process accounting. If process accounting is enabled, an accounting record will be written on an accounting file for each process that terminates. Termination can be caused by one of two things: an **exit()** call or a signal; see **exit**(2) and **sigvec**(2). The effective user ID of the calling process must be super-user to use this call.

*path* points to a path name naming the accounting file. The accounting file format is given in **acct**(5).

The accounting routine is enabled if *path* is not a NULL pointer and no errors occur during the system call. It is disabled if *path* is a NULL pointer and no errors occur during the system call.

If accounting is already turned on, and a successful **acct()** call is made with a non-NULL *path*, all subsequent accounting records will be written to the new accounting file.

## RETURN VALUE

The value −1 is returned if an error occurs, and external variable **errno** is set to indicate the cause of the error. Otherwise the value 0 is returned.

## ERRORS

**acct()** will fail if one of the following is true:

| | |
|---|---|
| EPERM | The caller is not the super-user. |
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| EINVAL | Support for accounting was not configured into the system. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The named file does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| EACCES | The file referred to by *path* is not a regular file. |
| ELOOP | Too many symbolic links were encountered in translating the path name. |
| EROFS | The named file resides on a read-only file system. |
| EFAULT | *path* points outside the process's allocated address space. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

## SEE ALSO

**exit**(2), **sigvec**(2), **acct**(5), **sa**(8)

## BUGS

No accounting is produced for programs running when a crash occurs. In particular non-terminating programs are never accounted for.

## NOTES

Accounting is automatically disabled when the file system the accounting file resides on runs out of space; it is enabled when space once again becomes available.

## NAME

adjtime – correct the time to allow synchronization of the system clock

## SYNOPSIS

```
#include <sys/time.h>

int adjtime(delta, olddelta)
struct timeval *delta;
struct timeval *olddelta;
```

## DESCRIPTION

**adjtime()** adjusts the system's notion of the current time, as returned by **gettimeofday**(2), advancing or retarding it by the amount of time specified in the **struct timeval** pointed to by *delta*.

The adjustment is effected by speeding up (if that amount of time is positive) or slowing down (if that amount of time is negative) the system's clock by some small percentage, generally a fraction of one percent. Thus, the time is always a monotonically increasing function. A time correction from an earlier call to **adjtime()** may not be finished when **adjtime()** is called again. If *olddelta* is not a NULL pointer, then the structure it points to will contain, upon return, the number of microseconds still to be corrected from the earlier call. If *olddelta* is a NULL pointer, the corresponding information will not be returned.

This call may be used in time servers that synchronize the clocks of computers in a local area network. Such time servers would slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time.

Only the super-user may adjust the time of day.

The adjustment value will be silently rounded to the resolution of the system clock.

## RETURN

A 0 return value indicates that the call succeeded. A −1 return value indicates an error occurred, and in this case an error code is stored into the global variable **errno**.

## ERRORS

The following error codes may be set in **errno**:

| | |
|---|---|
| EFAULT | *delta* or *olddelta* points outside the process's allocated address space, or *olddelta* points to a region of the process' allocated address space that is not writable. |
| EPERM | The process's effective user ID is not that of the super-user. |

## SEE ALSO

**date**(1V), **gettimeofday**(2)

## NAME

audit – write a record to the audit log

## SYNOPSIS

```
#include <sys/label.h>
#include <sys/audit.h>

audit (record)
audit_record_t *record;
```

## DESCRIPTION

The **audit()** system call is used to write a record to the system audit log file. The data pointed to by *record* is written to the audit log file. The data should be a well-formed audit record as described by **audit.log**(5). The kernel sets the time stamp value in the record and performs a minimal check on the data before writing it to the audit log file.

Only the super-user may successfully execute this call.

## RETURN VALUE

If the call succeeds, a value of 0 is returned. If an error occurs, the value −1 is returned.

## ERRORS

| | |
|---|---|
| EPERM | The process's effective user ID is not super-user. |
| EINVAL | The length specified in the audit record is too short, or more than **MAXAUDITDATA**. |
| EFAULT | *Record* points outside the process's allocated address space. |

## SEE ALSO

**auditsvc**(2), **getauid**(2), **setuseraudit**(2), **audit_args**(3), **audit.log**(5), **auditd**(8)

## NAME

auditon – manipulate auditing

## SYNOPSIS

**#include <sys/label.h>**
**#include <sys/audit.h>**

**auditon (condition)**
**int condition;**

## DESCRIPTION

The **auditon()** system call sets system auditing to the requested *condition* if and only if the current state of auditing allows that transition. Legitimate values for *condition* are:

| | |
|---|---|
| AUC_UNSET | on/off has not been decided yet |
| AUC_AUDITING | auditing is to be done |
| AUC_NOAUDIT | auditing is not to be done |

The permitted transitions are:

Any condition may be changed back to itself.

AUC_UNSET may be changed to AUC_AUDITING or AUC_NOAUDIT.

AUC_AUDITING may be changed to AUC_NOAUDIT.

AUC_NOAUDIT may be changed to AUC_AUDITING.

Once changed, it is not possible to get back to AUC_UNSET.

Only the super-user may successfully execute this call.

## RETURN VALUE

If the call succeeds the old audit condition value is returned. If an error occurs, the value −1 is returned.

## ERRORS

| | |
|---|---|
| EPERM | Neither of the process's effective or real user ID is super-user. |
| EINVAL | The *condition* specified is outside the range of valid values, or the current condition precludes the requested change. |

## SEE ALSO

**audit**(2), **setuseraudit**(2)

**NAME**

auditsvc – write audit records to specified file descriptor

**SYNOPSIS**

**auditsvc(fd, limit)**
**int fd;**
**int limit;**

**DESCRIPTION**

The **auditsvc()** system call specifies the audit log file to the kernel. The kernel writes audit records to this file until an exceptional condition occurs and then the call returns. The parameter *fd* is a file descriptor that identifies the audit file. Programs should open this file for writing before calling **auditsvc**. The parameter *limit* specifies a value between 0 and 100, instructing **auditsvc()** to return when the percentage of free disk space on the audit filesystem drops below this limit. Thus, the invoking program can take action to avoid running out of disk space. The **auditsvc()** system call does not return until one of the following conditions occurs:

- The process receives a signal that is not blocked or ignored.
- An error is encountered writing to the audit log file.
- The minimum free space (as specified by *limit*), has been reached.

Only processes with a real or effective user ID of super-user may execute this call successfully.

**RETURN VALUE**

This call only returns on an error.

**ERRORS**

| | |
|---|---|
| EPERM | The process's effective or real user ID is not super-user. |
| EBUSY | A second process attempted to perform this call. |
| EBADF | *d* is not a valid descriptor open for writing. |
| EPIPE | An attempt is made to write to a pipe that is not open for reading by any process (or to a socket of type SOCK_STREAM that is connected to a peer socket.) Note: an attempted write of this kind will also cause you to receive a **SIGPIPE** signal from the kernel. If you've not made a special provision to catch or ignore this signal, your process will die. |
| EFBIG | An attempt was made to write a file that exceeds the process's file size limit or the maximum file size. |
| EINTR | The call is forced to terminate prematurely due to the arrival of a signal whose **SV_INTERRUPT** bit in **sv_flags** is set (see **sigvec**(2)). **signal**(3V), in the System V compatibility library, sets this bit for any signal it catches. |
| ENOSPC | There is no free space remaining on the file system containing the file. |
| EDQUOT | The user's quota of disk blocks on the file system containing the file has been exhausted. |
| EDQUOT | Audit filesystem space is below the specified limit. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ENXIO | A hangup occurred on the *stream* being written to. |
| EWOULDBLOCK | The file was marked for 4.2BSD-style non-blocking I/O, and no data could be written immediately. |
| EAGAIN | The descriptor referred to a *stream*, was marked for System V-style non-blocking I/O, and no data could be written immediately. |
| EBUSY | A second process attempted to perform this call. |

**SEE ALSO**

**audit**(2), **sigvec**(2), **signal**(3V), **audit.log**(5), **auditd**(8)

## NAME

bind – bind a name to a socket

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

bind(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;
```

## DESCRIPTION

**bind()** assigns a name to an unnamed socket. When a socket is created with **socket**(2) it exists in a name space (address family) but has no name assigned. **bind()** requests that the name pointed to by *name* be assigned to the socket.

### NOTES

Binding a name in the UNIX domain creates a socket in the file system that must be deleted by the caller when it is no longer needed (using **unlink**(2)).

The rules used in name binding vary between communication domains. Consult the manual entries in section 4 for detailed information.

## RETURN VALUE

If the bind is successful, a 0 value is returned. A return value of −1 indicates an error, which is further specified in the global **errno**.

## ERRORS

The **bind()** call will fail if:

| | |
|---|---|
| EBADF | *s* is not a valid descriptor. |
| ENOTSOCK | *s* is a descriptor for a file, not a socket. |
| EADDRNOTAVAIL | The specified address is not available from the local machine. |
| EADDRINUSE | The specified address is already in use. |
| EINVAL | *namelen* is not the size of a valid address for the specified address family. |
| EINVAL | The socket is already bound to an address. |
| EACCES | The requested address is protected, and the current user has inadequate permission to access it. |
| EFAULT | The *name* parameter is not in a valid part of the user address space. |

The following errors are specific to binding names in the UNIX domain.

| | |
|---|---|
| ENOTDIR | A component of the path prefix of the path name in *name* is not a directory. |
| ENAMETOOLONG | The length of a component of the path name in *name* exceeds 255 characters, or the length of the path name in *name* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of the path name in *name* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of the path name in *name*. |
| ELOOP | Too many symbolic links were encountered in translating the path name in *name*. |
| EIO | An I/O error occurred while making the directory entry or allocating the inode. |
| EROFS | The inode would reside on a read-only file system. |
| EISDIR | A null path name was specified. |

**SEE ALSO**

**connect**(2), **getsockname**(2), **listen**(2), **socket**(2), **unlink**(2)

## NAME

brk, sbrk – change data segment size

## SYNOPSIS

**#include <sys/types.h>**

**int brk(addr)**
**caddr_t addr;**

**caddr_t sbrk(incr)**
**int incr;**

## DESCRIPTION

**brk()** sets the system's idea of the lowest data segment location not used by the program (called the *break*) to *addr* (rounded up to the next multiple of the system's page size).

In the alternate function **sbrk()**, *incr* more bytes are added to the program's data space and a pointer to the start of the new area is returned.

When a program begins execution using **execve()** the break is set at the highest location defined by the program and data storage areas.

The **getrlimit**(2) system call may be used to determine the maximum permissible size of the *data* segment; it will not be possible to set the break beyond the **rlim_max** value returned from a call to **getrlimit()**, that is to say, "**etext** + **rlim.rlim_max**." (See **end**(3) for the definition of **etext**.)

## RETURN VALUE

**brk()** returns 0 on success, while **sbrk()** returns the old break value. If the break cannot be set, **brk()** returns –1; **sbrk()** returns **(caddr_t)** –1 on error.

## ERRORS

**brk()** and **sbrk()** will fail and no additional memory will be allocated if one of the following are true:

| | |
|---|---|
| ENOMEM | The data segment size limit, as set by **setrlimit** (see **getrlimit**(2)), would be exceeded. |
| ENOMEM | The maximum possible size of a data segment (compiled into the system) would be exceeded. |
| ENOMEM | Insufficient space exists in the swap area to support the expansion. |
| ENOMEM | Out of address space; the new break value would extend into an area of the address space defined by some previously established mapping (see **mmap**(2). |

## SEE ALSO

**execve**(2), **mmap**(2), **getrlimit**(2), **malloc**(3), **end**(3)

## BUGS

Setting the break may fail due to a temporary lack of swap space. It is not possible to distinguish this from a failure caused by exceeding the maximum size of the data segment without consulting **getrlimit()**.

NAME

chdir – change current working directory

SYNOPSIS

**int chdir (path)**
**char *path;**

**int fchdir (fd)**
**int fd;**

DESCRIPTION

**chdir( )** and **fchdir** cause a directory to become the current working directory, that is, the starting point for pathnames not beginning with '/'.

In order for a directory to become the current directory, a process must have execute (search) access to the directory.

The *path* argument to **chdir( )** points to the pathname of a directory. The *fd* argument to **fchdir** is the open file descriptor of a directory.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

WARNING

**fchdir** is provided as a performance enhancement and is guaranteed to fail under certain conditions. In particular, if auditing is active the call will never succeed, and EINVAL will be returned. Applications which use this system call must be coded to detect this failure and switch to using **chdir( )** from that point on.

## NAME

chmod, fchmod – change mode of file

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int chmod(path, mode)
char *path;
int mode;

int fchmod(fd, mode)
int fd, mode;
```

## DESCRIPTION

The file whose name is given by *path* or referenced by the descriptor *fd* has its mode changed to *mode*. Modes are constructed by ORing together some combination of the following:

| | |
|---|---|
| **S_ISUID** | **04000 set user ID on execution** |
| **S_ISGID** | **02000 set group ID on execution** |
| **S_ISVTX** | **01000 save text image after execution (sticky bit)** |
| **S_IREAD** | **00400 read by owner** |
| **S_IWRITE** | **00200 write by owner** |
| **S_IEXEC** | **00100 execute (search on directory) by owner** |
| | **00070 read, write, execute (search) by group** |
| | **00007 read, write, execute (search) by others** |

These bit patterns are defined in **<sys/stat.h>**.

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user and the process attempts to set the set group ID bit on a file owned by a group which is not in its group access list, mode bit 02000 (set group ID on execution) is cleared.

If mode bit 01000 is set on a directory, an unprivileged user may not delete or rename files of other users in that directory.

If a user other than the super-user writes to a file, the set user ID and set group ID bits are turned off. This makes the system somewhat more secure by protecting set-user-ID (**set-group-ID**) files from remaining set-user-ID (**set-group-ID**) if they are modified, at the expense of a degree of compatibility.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**chmod()** will fail and the file mode will be unchanged if:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The file referred to by *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EPERM | The effective user ID does not match the owner of the file and the effective user ID is not the super-user. |
| EINVAL | *fd* refers to a socket, not to a file. |

EROFS The file referred to by *path* resides on a read-only file system.

EFAULT *path* points outside the process's allocated address space.

EIO An I/O error occurred while reading from or writing to the file system.

**fchmod()** will fail if:

EBADF The descriptor is not valid.

EROFS The file referred to by *fd* resides on a read-only file system.

EPERM The effective user ID does not match the owner of the file and the effective user ID is not the super-user.

EIO An I/O error occurred while reading from or writing to the file system.

**FILES**

**/usr/include/sys/stat.h**

**SEE ALSO**

**chown**(2), **open**(2V), **stat**(2), **sticky**(8)

## NAME

chown, fchown – change owner and group of a file

## SYNOPSIS

**int chown(path, owner, group)**
**char *path;**
**int owner, group;**

**int fchown(fd, owner, group)**
**int fd, owner, group;**

## DESCRIPTION

The file that is named by *path* or referenced by *fd* has its *owner* and *group* changed as specified. Only the super-user may change the owner of the file, because if users were able to give files away, they could defeat the file-space accounting procedures. The owner of the file may change the group to a group of which he is a member; the super-user may change the group arbitrarily.

**fchown()** is particularly useful when used in conjunction with the file locking primitives (see **flock**(2)).

If *owner* or *group* is specified as −1, the corresponding ID of the file is not changed.

If a process whose effective user ID is not super-user successfully changes the group ID of a file, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

If the final component of *path* is a symbolic link, the ownership and group of the symbolic link is changed, not the ownership and group of the file or directory to which it points.

## RETURN VALUE

Zero is returned if the operation was successful; −1 is returned, and a more specific error code is placed in the global variable **errno**, if an error occurs.

## ERRORS

**chown()** will fail and the file will be unchanged if:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The file referred to by *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EPERM | The user ID specified by *owner* is not the current owner ID of the file, or the group ID specified by *group* is not the current group ID of the file and is not in the process' group access list, and the effective user ID is not the super-user. |
| EROFS | The file referred to by *path* resides on a read-only file system. |
| EFAULT | *path* points outside the process's allocated address space. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

**fchown()** will fail if:

EBADF *fd* does not refer to a valid descriptor.

EINVAL
*fd* refers to a socket, not a file.

EPERM The user ID specified by *owner* is not the current owner ID of the file, or the group ID specified by *group* is not the current group ID of the file and is not in the group access list, and the effective user ID is not the super-user.

EROFS The file referred to by *fd* resides on a read-only file system.

EIO An I/O error occurred while reading from or writing to the file system.

**SEE ALSO**

**chmod**(2), **flock**(2)

## NAME

chroot – change root directory

## SYNOPSIS

**int chroot (dirname)**
**char *dirname;**

**int fchroot (fd)**
**int fd;**

## DESCRIPTION

**chroot()** and **fchroot** cause a directory to become the root directory, the starting point for path names beginning with '/'. The current working directory is unaffected by this call. This root directory setting is inherited across **execve**(2) and by all children of this process created with **fork**(2) calls.

In order for a directory to become the root directory a process must have execute (search) access to the directory and either the effective user ID of the process must be super-user or the target directory must be the system root or a loop-back mount of the system root (see **lofs**(4S)). **fchroot** is further restricted in that while it is always possible to change to the system root using this call, it is not guaranteed to succeed in any other case, even should *fd* be in all respects valid.

The *dirname* argument to **chroot()** points to a path name of a directory. The *fd* argument to **fchroot** is the open file descriptor of the directory which is to become the root.

The .. entry in the root directory is interpreted to mean the root directory itself. Thus, .. cannot be used to access files outside the subtree rooted at the root directory. Instead, **fchroot** can be used to set the root back to a directory which was opened before the root directory was changed.

## WARNING

The only use of **fchroot** that is appropriate is to change back to the system root. While it may succeed in some other cases, it is guaranteed to fail if auditing is enabled. Super-user processes are not exempt from this limitation.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate an error.

## ERRORS

**chroot()** will fail and the root directory will be unchanged if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *dirname* is not a directory. |
| ENOTDIR | The file referred to by *dirname* is not a directory. |
| EINVAL | **fchroot** attempted to change to a directory which is not the system root and external circumstances, such as auditing, do not allow this. |
| ENAMETOOLONG | The length of a component of *dirname* exceeds 255 characters, or the length of *dirname* exceeds 1023 characters. |
| ENOENT | The directory referred to by *dirname* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *dirname*. |
| EACCES | Search permission is denied for the directory referred to by *dirname*. |
| ELOOP | Too many symbolic links were encountered in translating *dirname*. |
| EPERM | The effective user ID is not super-user. |
| EFAULT | *dirname* points outside the process's allocated address space. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EBADF | The descriptor is not valid. |

**SEE ALSO**

**chdir**(2), **execve**(2), **fork**(2), **lofs**(4S)

## NAME

close – delete a descriptor

## SYNOPSIS

**int close (des)**
**int des;**

## DESCRIPTION

The **close( )** call deletes a descriptor from the per-process object reference table. If this is the last reference to the underlying object, then it will be deactivated. For example, on the last close of a file the current *seek* pointer associated with the file is lost; on the last close of a **socket**(2) associated naming information and queued data are discarded; on the last close of a file holding an advisory lock the lock is released (see **flock**(2) for further information).

A close of all of a process's descriptors is automatic on **exit**, but since there is a limit on the number of active descriptors per process, **close( )** is necessary for programs that deal with many descriptors.

When a process forks (see **fork**(2)), all descriptors for the new child process reference the same objects as they did in the parent before the fork. If a new process is then to be run using **execve**(2), the process would normally inherit these descriptors. Most of the descriptors can be rearranged with **dup**(2) or deleted with **close( )** before the **execve( )** is attempted, but if some of these descriptors will still be needed if the **execve( )** fails, it is necessary to arrange for them to be closed if the **execve( )** succeeds. The **fcntl**(2V) operation **F_SETFD** can be used to arrange that a descriptor will be closed after a successful **execve**, or to restore the default behavior, which is to not close the descriptor.

If a STREAMS (see **intro**(2)) file is closed, and the calling process had previously registered to receive a SIGPOLL signal (see **sigvec**(2)) for events associated with that file (see **I_SETSIG** in **streamio**(4)), the calling process will be unregistered for events associated with the file. The last **close( )** for a stream causes that stream to be dismantled. If the descriptor is not marked for no-delay mode and there have been no signals posted for the stream, **close( )** waits up to 15 seconds, for each module and driver, for any output to drain before dismantling the stream. If the descriptor is marked for no-delay mode or if there are any pending signals, **close( )** does not wait for output to drain, and dismantles the stream immediately.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and the global integer variable **errno** is set to indicate the error.

## ERRORS

**close( )** will fail if:

| | |
|---|---|
| EBADF | *d* is not an active descriptor. |
| EINTR | A signal was caught before the close completed. |

## SEE ALSO

**accept**(2), **dup**(2), **execve**(2), **fcntl**(2V), **flock**(2), **intro**(2), **open**(2V), **pipe**(2), **sigvec**(2), **socket**(2), **socketpair**(2), **streamio**(4)

## NAME

connect – initiate a connection on a socket

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

connect(s, name, namelen)
int s;
struct sockaddr *name;
int namelen;
```

## DESCRIPTION

The parameter *s* is a socket. If it is of type **SOCK_DGRAM**, then this call specifies the peer with which the socket is to be associated; this address is that to which datagrams are to be sent, and the only address from which datagrams are to be received. If it is of type **SOCK_STREAM**, then this call attempts to make a connection to another socket. The other socket is specified by *name* which is an address in the communications space of the socket. Each communications space interprets the *name* parameter in its own way. Generally, stream sockets may successfully **connect()** only once; datagram sockets may use **connect()** multiple times to change their association. Datagram sockets may dissolve the association by connecting to an invalid address, such as a null address.

## RETURN VALUE

If the connection or binding succeeds, then 0 is returned. Otherwise a −1 is returned, and a more specific error code is stored in **errno**.

## ERRORS

The call fails if:

| | |
|---|---|
| EBADF | *s* is not a valid descriptor. |
| ENOTSOCK | *s* is a descriptor for a file, not a socket. |
| EINVAL | *namelen* is not the size of a valid address for the specified address family. |
| EADDRNOTAVAIL | The specified address is not available on the remote machine. |
| EAFNOSUPPORT | Addresses in the specified address family cannot be used with this socket. |
| EISCONN | The socket is already connected. |
| ETIMEDOUT | Connection establishment timed out without establishing a connection. |
| ECONNREFUSED | The attempt to connect was forcefully rejected. The calling program should **close**(2) the socket descriptor, and issue another **socket**(2) call to obtain a new descriptor before attempting another **connect**(2) call. |
| ENETUNREACH | The network is not reachable from this host. |
| EADDRINUSE | The address is already in use. |
| EFAULT | The *name* parameter specifies an area outside the process address space. |
| EINPROGRESS | The socket is non-blocking and the connection cannot be completed immediately. It is possible to **select**(2) for completion by selecting the socket for writing. |
| EALREADY | The socket is non-blocking and a previous connection attempt has not yet been completed. |
| EINTR | The connection attempt was interrupted before any data arrived by the delivery of a signal. |

The following errors are specific to connecting names in the UNIX domain. These errors may not apply in future versions of the UNIX IPC domain.

| | |
|---|---|
| ENOTDIR | A component of the path prefix of the path name in *name* is not a directory. |

| | |
|---|---|
| ENAMETOOLONG | The length of a component of the path name in *name* exceeds 255 characters, or the length of the entire path name in *name* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of the path name in *name* does not exist. |
| ENOENT | The socket referred to by the path name in *name* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of the path name in *name*. |
| ELOOP | Too many symbolic links were encountered in translating the path name in *name*. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ENOTSOCK | The file referred to by *name* is not a socket. |
| EPROTOTYPE | The file referred to by *name* is a socket of a type other than the type of *s* (e.g., *s* is a **SOCK_DGRAM** socket, while *name* refers to a **SOCK_STREAM** socket). |

**SEE ALSO**

**accept**(2), **close**(2), **connect**(2), **getsockname**(2), **select**(2), **socket**(2)

## NAME

creat – create a new file

## SYNOPSIS

**int creat(name, mode)**
**char *name;**
**int mode;**

## DESCRIPTION

This interface is made obsolete by **open**(2V).

**creat**( ) creates a new ordinary file or prepares to rewrite an existing file named by the path name pointed to by *name*. If the file did not exist, it is given mode *mode*, as modified by the process's mode mask (see **umask**(2)). Also see **chmod**(2) for the construction of the *mode* argument.

If the file exists, its mode and owner remain unchanged, but it is truncated to 0 length. Otherwise, the file's owner ID is set to the effective user ID of the process.

The file's group ID is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2)) and the set-gid bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

The low-order 12 bits of the file mode are set to the value of *mode*, modified as follows:

- All bits set in the process's file mode creation mask are cleared. See **umask**(2).
- The "save text image after execution" (sticky) bit of the mode is cleared. See **chmod**(2).
- The "set group ID on execution" bit of the mode is cleared if the effective user ID of the process is not super-user and the process is not a member of the group of the created file.

Upon successful completion, the file descriptor is returned and the file is open for writing, even if the mode does not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across **execve**(2) system calls. See **fcntl**(2V).

### NOTES

The *mode* given is arbitrary; it need not allow writing. This feature has been used in the past by programs to construct a simple exclusive locking mechanism. It is replaced by the **O_EXCL** open mode, or **flock**(2) facility.

## RETURN VALUE

The value −1 is returned if an error occurs. Otherwise, the call returns a non-negative descriptor which only permits writing.

## ERRORS

**creat**( ) will fail and the file will not be created or truncated if one of the following occur:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *name* is not a directory. |
| ENAMETOOLONG | The length of a component of *name* exceeds 255 characters, or the length of *name* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of *name* does not exist. |
| ELOOP | Too many symbolic links were encountered in translating *name*. |
| EACCES | Search permission is denied for a component of the path prefix of *name*. |
| EACCES | The file referred to by *name* does not exist and the directory in which it is to be created is not writable. |
| EACCES | The file referred to by *name* exists, but it is unwritable. |
| EISDIR | The file referred to by *name* is a directory. |

| | |
|---|---|
| EMFILE | There are already too many files open. |
| ENFILE | The system file table is full. |
| ENOSPC | The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory. |
| ENOSPC | There are no free inodes on the file system on which the file is being created. |
| EDQUOT | The directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted. |
| EDQUOT | The user's quota of inodes on the file system on which the file is being created has been exhausted. |
| EROFS | The file referred to by *name* resides, or would reside, on a read-only file system. |
| ENXIO | The file is a character special or block special file, and the associated device does not exist. |
| EIO | An I/O error occurred while making the directory entry or allocating the inode. |
| EFAULT | *name* points outside the process's allocated address space. |
| EOPNOTSUPP | The file was a socket (not currently implemented). |

**SEE ALSO**

**close**(2), **chmod**(2), **execve**(2), **fcntl**(2V), **flock**(2), **mount**(2), **open**(2V), **write**(2V), **umask**(2)

## NAME

dup, dup2 – duplicate a descriptor

## SYNOPSIS

**int dup(oldd)**
**int oldd;**

**int dup2(oldd, newd)**
**int oldd, newd;**

## DESCRIPTION

**dup()** duplicates an existing object descriptor. The argument *oldd* is a small non-negative integer index in the per-process descriptor table. The value must be less than the size of the table, which is returned by **getdtablesize**(2). The new descriptor returned by the call is the lowest numbered descriptor that is not currently in use by the process.

In the second form of the call, the value of the new descriptor desired is specified. If that descriptor is already in use, the descriptor is first deallocated as if a **close**(2) call had been done first.

The new descriptor has the following in common with the original:

It refers to the same object that the old descriptor referred to.

It uses the same file pointer as the old descriptor. (that is, both file descriptors share one file pointer).

It has the same access mode (read, write or read/write) as the old descriptor.

Thus if *newd* and *oldd* are duplicate references to an open file, **read**(2V), **write**(2V) and **lseek**(2) calls all move a single pointer into the file, and append mode, non-blocking I/O and asynchronous I/O options are shared between the references. If a separate pointer into the file is desired, a different object reference to the file must be obtained by issuing an additional **open**(2V) call. The close-on-exec flag on the new file descriptor is unset.

The new file descriptor is set to remain open across **exec** system calls. See **fcntl**(2V).

## RETURN VALUE

The value −1 is returned if an error occurs in either call. The external variable **errno** indicates the cause of the error.

## ERRORS

**dup()** and **dup2()** fail if:

| | |
|---|---|
| EBADF | *oldd* or *newd* is not a valid active descriptor. |
| EMFILE | Too many descriptors are active. |

## SEE ALSO

**accept**(2), **close**(2), **fcntl**(2V), **getdtablesize**(2), **lseek**(2), **open**(2V), **pipe**(2), **read**(2V), **socket**(2), **socketpair**(2), **write**(2V)

## NAME

execve – execute a file

## SYNOPSIS

**int execve(path, argv, envp)**
**char *path, **argv, **envp;**

## DESCRIPTION

**execve()** transforms the calling process into a new process. The new process is constructed from an ordinary file, whose name is pointed to by *path*, called the *new*process*file*. This file is either an executable object file, or a file of data for an interpreter. An executable object file consists of an identifying header, followed by pages of data representing the initial program (text) and initialized data pages. Additional pages may be specified by the header to be initialized with zero data. See **a.out**(5).

An interpreter file begins with a line of the form '**#!** *interpreter* [*arg*]'. When an interpreter file is **execve**'d, the system **execve**'s the specified *interpreter*. If the optional *arg* is specified, it becomes the first argument to the *interpreter*, and the name of the originally **execve**'d file becomes the second argument; otherwise, the name of the originally **execve**'d file becomes the first argument. The original argument are shifted over to become the subsequent arguments. The zeroth argument, normally the name of the **execve**'d file, is left unchanged.

There can be no return from a successful **execve()** because the calling core image is lost. This is the mechanism whereby different process images become active.

The argument *argv* is a pointer to a NULL-terminated array of character pointers to null-terminated character strings. These strings constitute the argument list to be made available to the new process. By convention, at least one argument must be present in this array, and the first element of this array should be the name of the executed program (that is, the last component of *path*).

The argument *envp* is also a pointer to a NULL-terminated array of character pointers to null-terminated strings. These strings pass information to the new process which are not directly arguments to the command (see **environ**(5V)).

Descriptors open in the calling process remain open in the new process, except for those for which the close-on-exec flag is set (see **close**(2) and **fcntl**(2V)). Descriptors which remain open are unaffected by **execve**.

Ignored signals remain ignored across an **execve**, but signals that are caught are reset to their default values. Blocked signals remain blocked regardless of changes to the signal action. The signal stack is reset to be undefined (see **sigvec**(2) for more information).

Each process has a *real* user ID and group ID and an *effective* user ID and group ID. The *real* ID identifies the person using the system; the *effective* ID determines their access privileges. **execve()** changes the effective user or group ID to the owner or group of the executed file if the file has the ''set-user-ID'' or ''set-group-ID'' modes. The *real* user ID and group ID are not affected.

The shared memory segments attached to the calling process will not be attached to the new process (see **shmop**(2)).

Profiling is disabled for the new process; see **profil**(2).

The new process also inherits the following attributes from the calling process:

| | |
|---|---|
| process ID | see **getpid**(2) |
| parent process ID | see **getpid**(2) |
| process group ID | see **setpgrp**(2V) |
| access groups | see **getgroups**(2) |
| semadj values | see**semop**(2) |
| working directory | see **chdir**(2) |
| root directory | see **chroot**(2) |
| control terminal | see **termio**(4) |

| | |
|---|---|
| trace flag | see **ptrace**(2) request 0) |
| resource usages | see **getrusage**(2) |
| interval timers | see **getitimer**(2) |
| resource limits | see **getrlimit**(2) |
| file mode mask | see **umask**(2) |
| signal mask | see **sigvec**(2), **sigsetmask**(2) |

When the executed program begins, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where *argc* is the number of elements in *argv* (the "arg count") and *argv* points to the array of character pointers to the arguments themselves.

*envp* is a pointer to an array of strings that constitute the *environment* of the process. A pointer to this array is also stored in the global variable **environ**. Each string consists of a name, an "=", and a null-terminated value. The array of pointers is terminated by a null pointer. The shell **sh**(1) passes an environment entry for each global shell variable defined when the program is called. See **environ**(5V) for some conventionally used names.

## RETURN VALUE

If **execve**( ) returns to the calling process an error has occurred; the return value will be −1 and the global variable **errno** will contain an error code.

## ERRORS

**execve**( ) will fail and return to the calling process if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of the new process file is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | One or more components of the path prefix of the new process file does not exist. |
| ENOENT | The new process file does not exist. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EACCES | Search permission is denied for a component of the new process file's path prefix. |
| EACCES | The new process file is not an ordinary file. |
| EACCES | Execute permission is denied for the new process file. |
| ENOEXEC | The new process file has the appropriate access permission, but has an invalid magic number in its header. |
| ENOMEM | The new process file requires more virtual memory than is allowed by the imposed maximum (**getrlimit**(2)). |
| E2BIG | The number of bytes in the new process file's argument list is larger than the system-imposed limit. The limit in the system as released is 1,048,576 bytes (**NCARGS** in **<sys/param.h>**). |
| EFAULT | The new process file is not as long as indicated by the size values in its header. |
| EFAULT | *path*, *argv*, or *envp* points to an illegal address. |
| EIO | An I/O error occurred while reading from the file system. |

## CAVEATS

If a program is **setuid**( ) to a non-super-user, but is executed when the real user ID is super-user, then the program has some of the powers of a super-user as well.

**SEE ALSO**

**sh**(1), **chdir**(2), **chroot**(2), **close**(2), **exit**(2), **fcntl**(2V), **fork**(2), **getgroups**(2), **getitimer**(2), **getpid**(2), **getrlimit**(2), **getrusage**(2), **profil**(2), **ptrace**(2), **semop**(2), **setpgrp**(2V), **shmop**(2), **sigvec**(2), **execl**(3), **termio**(4), **a.out**(5), **environ**(5V)

**NAME**

_exit – terminate a process

**SYNOPSIS**

**_exit(status)**
**int status;**

**DESCRIPTION**

**_exit()** terminates a process with the following consequences:

All of the descriptors open in the calling process are closed. This may entail delays, for example, waiting for output to drain; a process in this state may not be killed, as it is already dying.

If the parent process of the calling process is executing a **wait()** or is interested in the **SIGCHLD** signal, then it is notified of the calling process's termination and the low-order eight bits of *status* are made available to it; see **wait**(2).

The parent process ID of all of the calling process's existing child processes are also set to 1. This means that the initialization process (see **intro**(2)) inherits each of these processes as well. Any stopped children are restarted with a hangup signal (**SIGHUP**).

Each attached shared memory segment is detached and the value of **shm_nattach** in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the calling process has set a *semadj* value (see **semop**(2)), that *semadj* value is added to the *semval* of the specified semaphore.

If process accounting is enabled (see **acct**(2)), an accounting record is written to the accounting file.

Most C programs will call the library routine **exit**(3) which performs cleanup actions in the standard I/O library before calling **_exit**.

**RETURN VALUE**

This call never returns.

**SEE ALSO**

**acct**(2), **fork**(2), **intro**(2), **semop**(2), **wait**(2), **exit**(3)

NAME

fcntl – file control

SYNOPSIS

**#include <fcntl.h>**

**int fcntl (des, cmd, arg)**
**int des, cmd, arg;**

DESCRIPTION

**fcntl()** performs a variety of functions on open descriptors. The argument *des* is an open descriptor to be operated on by *cmd* as follows:

**F_DUPFD** Return a new descriptor as follows:

Lowest numbered available descriptor greater than or equal to *arg*.

Refers to the same object as the original descriptor.

New descriptor shares the same file pointer if the object was a file (that is, both descriptors share one file pointer).

Same access mode (read, write or read/write).

Same descriptor status flags (both descriptors share the same descriptor status flags).

The close-on-exec flag associated with the new descriptor is set to remain open across **execve**(2) system calls.

**F_GETFD** Get the close-on-exec flag associated with the descriptor *des*. If the low-order bit is **0**, the file will remain open across **execve**, otherwise the file will be closed upon execution of **execve**.

**F_SETFD** Set the close-on-exec flag associated with *des* to the low order bit of *arg* (**0** or **1** as above). Note: this flag is a per-process and per-descriptor flag; setting or clearing it for a particular descriptor will not affect the flag on descriptors copied from it by a **dup**(2) or **F_DUPFD** operation, nor will it affect the flag on other processes instances of that descriptor.

**F_GETFL** Get descriptor status flags (see **fcntl**(5) for their definitions).

**F_SETFL** Set descriptor status flags (see **fcntl**(5) for their definitions). Only the following flags can have their values changed: **O_APPEND**, **O_SYNC**, and **O_NDELAY**, and the **FASYNC**, **FNDELAY**, and **FNBIO** flags defined in **<sys/file.h>**.

In the 4.2BSD environment, the **O_NDELAY** and **FNDELAY** flags are the same flag; in the System V environment, the **O_NDELAY** and **FNBIO** flags are the same flag. The **FNDELAY** and **FNBIO** flags may be used in either environment; the meaning of those flags is not dependent on the environment in which a program is built.

As the descriptor status flags are shared with descriptors copied from a given descriptor by a **dup**(2) or **F_DUPFD** operation, and by other processes instances of that descriptor, a **F_SETFL** operation will affect those other descriptors and other instances of the given descriptor as well. In addition, setting or clearing the **FNDELAY** flag on a descriptor will cause an **FIONBIO** **ioctl**(2) to be performed on the object referred to by that descriptor, setting or clearing non-blocking mode, and setting or clearing the **FASYNC** flag on a descriptor will cause an **FIOASYNC** **ioctl**(2) to be performed on the object referred to by that descriptor, setting or clearing asynchronous mode. Thus, all descriptors referring to that object will be affected.

**F_GETLK** Get a description of the first lock that would block the lock specified in the **flock()**

structure pointed to by *arg*. The information retrieved overwrites the information in the **flock()** structure. If no lock is found that would prevent this lock from being created, then the structure is passed back unchanged except for the lock type which will be set to F_UNLCK.

**F_SETLK** Set or clear an advisory record lock according to the **flock()** structure pointed to by *arg*. F_SETLK is used to establish shared (F_RDLCK) and exclusive (F_WRLCK) locks, or to remove either type of lock (F_UNLCK). If the specified lock cannot be applied, **fcntl()** will return with an error value of −1.

**F_SETLKW** This *cmd* is the same as F_SETLK except that if a shared or exclusive lock is blocked by other locks, the requesting process will sleep until the lock may be applied.

**F_GETOWN** Get the process ID or process group currently receiving SIGIO and SIGURG signals; process groups are returned as negative values.

**F_SETOWN** Set the process or process group to receive SIGIO and SIGURG signals; process groups are specified by supplying *arg* as negative, otherwise *arg* is interpreted as a process ID.

The SIGIO facilities are enabled by setting the FASYNC flag with F_SETFL.

NOTES

Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access (processes may still access files without using advisory locks, possibly resulting in inconsistencies).

The record locking mechanism allows two types of locks: shared locks (F_RDLCK) and exclusive locks (F_WRLCK). More than one process may hold a shared lock for a particular segment of a file at any given time, but multiple exclusive, or both shared and exclusive, locks may not exist simultaneously on any segment.

In order to claim a shared lock, the descriptor must have been opened with read access. The descriptor on which an exclusive lock is being placed must have been opened with write access.

A shared lock may be *upgraded* to an exclusive lock, and vice versa, simply by specifying the appropriate lock type with a *cmd* of F_SETLK or F_SETLKW; the previous lock will be released and the new lock applied (possibly after other processes have gained and released the lock).

If the *cmd* is F_SETLKW and the requested lock cannot be claimed immediately (for instance, another process holds an exclusive lock that partially or completely overlaps the current request) then the calling process will block until the lock may be acquired. Processes blocked awaiting a lock may be awakened by signals.

Care should be taken to avoid deadlock situations in applications in which multiple processes perform blocking locks on a set of common records.

The record that is to be locked or unlocked is described by the **flock()** structure, which is defined in **<fcntl.h>** and includes the following members:

```
short l_type;    /* F_RDLCK, F_WRLCK, or F_UNLCK */
short l_whence; /* flag to choose starting offset */
long  l_start;   /* relative offset, in bytes */
long  l_len;     /* length, in bytes; 0 means lock to EOF */
short l_pid;     /* returned with F_GETLK */
```

The **flock** structure describes the type (**l_type**), starting offset (**l_whence**), relative offset (**l_start**), and size (**l_len**) of the segment of the file to be affected. **l_whence** must be set to 0, 1, or 2 to indicate that the relative offset will be measured from the start of the file, current position, or end-of-file, respectively. The process id field (**l_pid**) is only used with the **F_GETLK** *cmd* to return the description of a lock held by another process.

Locks may start and extend beyond the current end-of-file, but may not be negative relative to the beginning of the file. A lock may be set to always extend to the end-of-file by setting **l_len** to zero (0). If such a lock also has **l_whence** and **l_start** set to zero (0), the entire file will be locked. Changing or unlocking a segment from the middle of a larger locked segment leaves two smaller segments at either end. Locking a segment that is already locked by the calling process causes the old lock type to be removed and the new lock type to take affect. All locks associated with a file for a given process are removed when the file is closed or the process terminates. Locks are not inherited by the child process in a **fork**(2) system call.

In order to maintain consistency in the network case, data must not be cached on client machines. For this reason, file buffering for an NFS file is turned off when the first lock is attempted on the file. Buffering will remain off as long as the file is open. Programs that do I/O buffering in the user address space, however, may have inconsistent results (the standard I/O package, for instance, is a common source of unexpected buffering).

The advisory record locking capabilities of **fcntl()** are implemented throughout the network by the **network lock daemon**; see **lockd** (8C). If the file server crashes and is rebooted, the lock daemon will attempt to recover all locks that were associated with that server. If a lock cannot be reclaimed, the process that held the lock will be issued a **SIGLOST** signal.

## RETURN VALUE

Upon successful completion, the value returned depends on *cmd* as follows:

**F_DUPFD** A new descriptor.
**F_GETFD** Value of flag (only the low-order bit is defined).
**F_GETFL** Value of flags.
**F_GETOWN**
Value of descriptor owner.
other Value other than −1. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**fcntl()** will fail if one or more of the following are true:

EBADF *des* is not a valid open descriptor.

EMFILE *cmd* is **F_DUPFD** and the maximum allowed number of descriptors are currently open.

EINVAL *cmd* is **F_DUPFD** and *arg* is negative or greater than the maximum allowable number (see **getdtablesize**(2)).

EFAULT *cmd* is **F_GETLK**, **F_SETLK**, or **F_SETLKW** and *arg* points to an invalid address.

EINVAL *cmd* is **F_GETLK**, **F_SETLK**, or **F_SETLKW** and the data *arg* points to is not valid.

EBADF *cmd* is **F_SETLK** or **F_SETLKW** and the process does not have the appropriate read or write permissions on the file.

EAGAIN *cmd* is **F_SETLK**, the lock type (**l_type**) is **F_RDLCK** (shared lock), and the segment of the file to be locked already has an exclusive lock held by another process. This error will also be returned if the lock type is **F_WRLCK** (exclusive lock) and another process already has the segment locked with either a shared or exclusive lock.

EINTR *cmd* is **F_SETLKW** and a signal interrupted the process while it was waiting for the lock to be granted.

ENOLCK *cmd* is **F_SETLK** or **F_SETLKW** and there are no more file lock entries available.

**SEE ALSO**

**close**(2), **execve**(2), **flock**(2), **fork**(2), **getdtablesize**(2), **ioctl**(2), **open**(2V), **sigvec**(2), **lockf**(3), **fcntl**(5), **lockd**(8C)

**BUGS**

File locks obtained through the **fcntl**() mechanism do not interact in any way with those acquired via **flock**(2). They do, however, work correctly with the exclusive locks claimed by **lockf**(3).

**F_GETLK** returns **F_UNLCK** if the requesting process holds the specified lock. Thus, there is no way for a process to determine if it is still holding a specific lock after catching a **SIGLOST** signal.

In a network environment, the value of **l_pid** returned by **F_GETLK** is next to useless.

NAME

flock – apply or remove an advisory lock on an open file

SYNOPSIS

```
#include <sys/file.h>

#define LOCK_SH     1     /* shared lock */
#define LOCK_EX     2     /* exclusive lock */
#define LOCK_NB     4     /* don't block when locking */
#define LOCK_UN     8     /* unlock */

flock(fd, operation)
int fd, operation;
```

DESCRIPTION

**flock()** applies or removes an *advisory* lock on the file associated with the file descriptor **fd**. A lock is applied by specifying an *operation* parameter that is the inclusive OR of **LOCK_SH** or **LOCK_EX** and, possibly, **LOCK_NB**. To unlock an existing lock, the *operation* should be **LOCK_UN**.

Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access (that is, processes may still access files without using advisory locks, possibly resulting in inconsistencies).

The locking mechanism allows two types of locks: *shared* locks and *exclusive* locks. More than one process may hold a shared lock for a file at any given time, but multiple exclusive, or both shared and exclusive, locks may not exist simultaneously on a file.

A shared lock may be *upgraded* to an exclusive lock, and vice versa, simply by specifying the appropriate lock type; the previous lock will be released and the new lock applied (possibly after other processes have gained and released the lock).

Requesting a lock on an object that is already locked normally causes the caller to block until the lock may be acquired. If **LOCK_NB** is included in *operation*, then this will not happen; instead the call will fail and the error EWOULDBLOCK will be returned.

NOTES

Locks are on files, not file descriptors. That is, file descriptors duplicated through **dup**(2) or **fork**(2) do not result in multiple instances of a lock, but rather multiple references to a single lock. If a process holding a lock on a file forks and the child explicitly unlocks the file, the parent will lose its lock.

Processes blocked awaiting a lock may be awakened by signals.

RETURN VALUE

Zero is returned on success, −1 on error, with an error code stored in **errno**.

ERRORS

The **flock()** call fails if:

| | |
|---|---|
| EWOULDBLOCK | The file is locked and the **LOCK_NB** option was specified. |
| EBADF | The argument **fd** is an invalid descriptor. |
| EOPNOTSUPP | The argument **fd** refers to an object other than a file. |

SEE ALSO

**close**(2), **dup**(2), **execve**(2), **fcntl**(2V), **fork**(2), **open**(2V), **lockf**(3), **lockd**(8C)

BUGS

Locks obtained through the **flock()** mechanism are known only within the system on which they were placed. Thus, multiple clients may successfully acquire exclusive locks on the same remote file. If this behavior is not explicitly desired, the **fcntl**(2V) or **lockf**(3) system calls should be used instead; these make use of the services of the **network lock manager** (see **lockd(8C)**).

## NAME

fork – create a new process

## SYNOPSIS

**int fork()**

## DESCRIPTION

**fork**( ) creates a new process. The new process (child process) is an exact copy of the calling process except for the following:

The child process has a unique process ID.

The child process has a different parent process ID (that is, the process ID of the parent process).

The child process has its own copy of the parent's descriptors. These descriptors reference the same underlying objects, so that, for instance, file pointers in file objects are shared between the child and the parent, so that an **lseek**(2) on a descriptor in the child process can affect a subsequent **read**(2V) or **write**(2V) by the parent. This descriptor copying is also used by the shell to establish standard input and output for newly created processes as well as to set up pipes.

All *semadj* values are cleared; see **semop**(2).

The child processes resource utilizations are set to 0; see **setrlimit**(2). The **it_value** and **it_interval** values for the **ITIMER_REAL** timer are reset to 0; see **getitimer**(2).

## RETURN VALUE

Upon successful completion, **fork**( ) returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of −1 is returned to the parent process, no child process is created, and the global variable **errno** is set to indicate the error.

## ERRORS

**fork**( ) will fail and no child process will be created if one or more of the following are true:

| | |
|---|---|
| EAGAIN | The system-imposed limit on the total number of processes under execution would be exceeded. This limit is determined when the system is generated. |
| EAGAIN | The system-imposed limit on the total number of processes under execution by a single user would be exceeded. This limit is determined when the system is generated. |
| ENOMEM | There is insufficient swap space for the new process. |

## SEE ALSO

**execve**(2), **getitimer**(2), **getrlimit**(2), **lseek**(2), **read**(2V), **semop**(2), **wait**(2), **write**(2V)

**NAME**

fsync – synchronize a file's in-core state with that on disk

**SYNOPSIS**

**int fsync(fd)**

**int fd;**

**DESCRIPTION**

**fsync**( ) moves all modified data and attributes of *fd* to a permanent storage device: all in-core modified copies of buffers for the associated file have been written to a disk when the call returns. Note: this is different than **sync**(8) which schedules disk I/O for all files (as though an **fsync**( ) had been done on all files) but returns before the I/O completes.

**fsync**( ) should be used by programs which require a file to be in a known state; for example, a program which contains a simple transaction facility might use it to ensure that all modifications to a file or files caused by a transaction were recorded on disk.

**RETURN VALUE**

A 0 value is returned on success. A –1 value indicates an error.

**ERRORS**

The **fsync**( ) fails if:

| | |
|---|---|
| EBADF | **fd** is not a valid descriptor. |
| EINVAL | *fd* refers to a socket, not a file. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

**SEE ALSO**

**cron**(8), **sync**(8)

**BUGS**

The current implementation of this call is expensive for large files.

**NAME**

getauid, setauid – get and set user audit identity

**SYNOPSIS**

**getauid()**

**setauid(auid)**
int auid;

**DESCRIPTION**

The **getauid()** system call returns the audit user ID for the current process. This value is initially set at login time and inherited by all child processes. This value does not change when the real/effective user IDs change, so it can be used to identify the logged-in user, even when running a setuid program. The audit user ID governs audit decisions for a process.

The **getauid()** system calls sets the audit user ID for the current process. Only the super-user may successfully execute these calls.

**RETURN VALUE**

The **getauid()** call returns the audit user ID of the current process on successful operation, and returns –1 for all errors.

The **getauid()** call returns 0 on successful operation, and –1 for all errors.

**ERRORS**

| | |
|---|---|
| EPERM | The process's effective user ID is not super-user. |
| EINVAL | The parameter *auid* is not a valid uid. |

**SEE ALSO**

**getuid**(2), **setuseraudit**(2), **audit**(8)

## NAME

getdents – gets directory entries in a filesystem independent format

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/dirent.h>

int getdents(fd, buf, nbytes)
int fd;
char *buf;
int nbytes;
```

## DESCRIPTION

**getdents**( ) attempts to put directory entries from the directory referenced by the file descriptor *fd* into the buffer pointed to by *buf*, in a filesystem independent format. Up to *nbytes* bytes of data will be transferred.

The data in the buffer is a series of **dirent** structures each containing the following entries:

```
off_t       d_off;
u_long      d_fileno;
u_short     d_reclen;
u_short     d_namlen;
char        d_name[MAXNAMLEN + 1];  /* see below */
```

The **d_off** entry contains a value which is interpretable only by the filesystem that generated it. It may be supplied as an offset to **lseek**(2) to find the entry following the current one in a directory. The **d_fileno** entry is a number which is unique for each distinct file in the filesystem. Files that are linked by hard links (see **link**(2)) have the same **d_fileno**. The **d_reclen** entry is the length, in bytes, of the directory record. The **d_name** entry contains a null terminated file name. The **d_namlen** entry specifies the length of the file name. Thus the actual size of **d_name** may vary from 1 to MAXNAMLEN+1.

The structures are not necessarily tightly packed. The **d_reclen** entry may be used as an offset from the beginning of a *dirent* structure to the next structure, if any.

Upon return, the actual number of bytes transferred is returned. The current position pointer associated with *fd* is set to point to the directory entry following the last one returned. The pointer is not necessarily incremented by the number of bytes returned by **getdents**. If the value returned is zero, the end of the directory has been reached. The current position pointer may be set and retrieved by *lseek(2)*. It is not safe to set the current position pointer to any value other than a value previously returned by **lseek**(2), or the value of a **d_off** entry in a **dirent** structure returned by **getdents**, or zero.

## RETURN VALUE

If successful, the number of bytes actually transferred is returned. Otherwise, a −1 is returned and the global variable **errno** is set to indicate the error.

## ERRORS

**getdents**( ) will fail if one or more of the following are true:

| | |
|---|---|
| EINVAL | *nbytes* is not large enough for one directory entry. |
| EBADF | *fd* is not a valid file descriptor open for reading. |
| ENOTDIR | The file referenced by *fd* is not a directory. |
| EFAULT | *buf* points outside the allocated address space. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EINTR | A read from a slow device was interrupted before any data arrived by the delivery of a signal. |

## SEE ALSO

**link**(2), **lseek**(2), **open**(2V), **directory**(3)

NOTES

It is strongly recommended, for portability reasons, that programs that deal with directory entries use the **directory**(3) interface rather than directly calling **getdents**.

## NAME

getdirentries – gets directory entries in a filesystem independent format

## SYNOPSIS

```
int getdirentries(fd, buf, nbytes, basep)
int fd;
char *buf;
int nbytes;
long *basep;
```

## DESCRIPTION

This system call is now obsolete. It is superseded by the **getdents**(2) system call, which returns directory entries in a new format specified in **<sys/dirent.h>**. The file, **<sys/dir.h>**, has also been modified to use the new directory entry format. Programs which currently call **getdirentries**( ) should be modified to use the new system call and the new include file **<sys/dirent.h>** or, preferably, to use the **directory**(3) library routines. The **getdirentries**( ) system call is retained in the current SunOS release only for purposes of backwards binary compatibility and will be removed in a future major release.

**getdirentries**( ) attempts to put directory entries from the directory referenced by the file descriptor **fd** into the buffer pointed to by *buf*, in a filesystem independent format. Up to *nbytes* bytes of data will be transferred. *nbytes* must be greater than or equal to the block size associated with the file, see **stat(2)**. Sizes less than this may cause errors on certain filesystems.

The data in the buffer is a series of structures each containing the following entries:

```
unsigned long  d_fileno;
unsigned short d_reclen;
unsigned short d_namlen;
char           d_name[MAXNAMELEN + 1];  /* see below */
```

The **d_fileno** entry is a number which is unique for each distinct file in the filesystem. Files that are linked by hard links (see **link**(2)) have the same **d_fileno**. The **d_reclen** entry is the length, in bytes, of the directory record. The **d_name** entry contains a null terminated file name. The **d_namlen** entry specifies the length of the file name. Thus the actual size of **d_name** may vary from 2 to **MAXNAMELEN+1**.

The structures are not necessarily tightly packed. The **d_reclen** entry may be used as an offset from the beginning of a **direct** structure to the next structure, if any.

Upon return, the actual number of bytes transferred is returned. The current position pointer associated with **fd** is set to point to the next block of entries. The pointer is not necessarily incremented by the number of bytes returned by **getdirentries**. If the value returned is zero, the end of the directory has been reached. The current position pointer may be set and retrieved by **lseek**(2). **getdirentries**( ) writes the position of the block read into the location pointed to by *basep*. It is not safe to set the current position pointer to any value other than a value previously returned by **lseek**(2) or a value previously returned in the location pointed to by *basep* or zero.

## RETURN VALUE

If successful, the number of bytes actually transferred is returned. Otherwise, a –1 is returned and the global variable **errno** is set to indicate the error.

## ERRORS

**getdirentries( )** will fail if one or more of the following are true:

| | |
|---|---|
| EBADF | *fd* is not a valid file descriptor open for reading. |
| EFAULT | Either *buf* or *basep* point outside the allocated address space. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EINTR | A read from a slow device was interrupted before any data arrived by the delivery of a signal. |

**SEE ALSO**

**getdents**(2), **link**(2), **lseek**(2), **open**(2V), **stat**(2), **directory**(3)

## NAME

getdomainname, setdomainname – get/set name of current domain

## SYNOPSIS

**int getdomainname(name, namelen)**
**char *name;**
**int namelen;**

**int setdomainname(name, namelen)**
**char *name;**
**int namelen;**

## DESCRIPTION

**getdomainname()** returns the name of the domain for the current processor, as previously set by **getdomainname**. The parameter *namelen* specifies the size of the array pointed to by *name*. The returned name is null-terminated unless insufficient space is provided.

**getdomainname()** sets the domain of the host machine to be *name*, which has length *namelen*. This call is restricted to the super-user and is normally used only when the system is bootstrapped.

The purpose of domains is to enable two distinct networks that may have host names in common to merge. Each network would be distinguished by having a different domain name. At the current time, only the Yellow Pages service makes use of domains.

## RETURN VALUE

If the call succeeds a value of 0 is returned. If the call fails, then a value of −1 is returned and an error code is placed in the global location **errno**.

## ERRORS

The following errors may be returned by these calls:

| | |
|---|---|
| EFAULT | The *name* parameter gave an invalid address. |
| EPERM | The caller was not the super-user. This error only applies to **setdomainname**. |

## BUGS

Domain names are limited to 255 characters.

**NAME**

getdtablesize – get descriptor table size

**SYNOPSIS**

**nds = getdtablesize( )**
**int nds;**

**DESCRIPTION**

Each process has a fixed size descriptor table, which is guaranteed to have at least 20 slots. The entries in the descriptor table are numbered with small integers starting at 0. The call **getdtablesize( )** returns the size of this table.

**SEE ALSO**

**close**(2), **dup**(2), **open**(2V)

**NAME**

getgid, getegid – get group identity

**SYNOPSIS**

```
gid = getgid()
int gid;

egid = getegid()
int egid;
```

**DESCRIPTION**

**getgid( )** returns the real group ID of the current process, **getegid( )** the effective group ID.

The real GID is specified at login time.

The effective GID is more transient, and determines additional access permission during execution of a "setGID" process, and it is for such processes that **getgid( )** is most useful.

**SEE ALSO**

**getuid**(2), **setregid**(2), **setuid**(3)

## NAME

getgroups, setgroups – get or set group access list

## SYNOPSIS

**#include <sys/param.h>**

**int getgroups(gidsetlen, gidset)**
**int gidsetlen, *gidset;**

**int setgroups(ngroups, gidset)**
**int ngroups, *gidset;**

## DESCRIPTION

### getgroups

**getgroups()** gets the current group access list of the user process and stores it in the array *gidset*. The parameter *gidsetlen* indicates the number of entries that may be placed in *gidset*. **getgroups()** returns the actual number of entries placed in the *gidset* array. No more than **NGROUPS**, as defined in **<sys/param.h>**, will ever be returned.

### setgroups

**setgroups()** sets the group access list of the current user process according to the array *gidset*. The parameter *ngroups* indicates the number of entries in the array and must be no more than **NGROUPS**, as defined in **<sys/param.h>**.

Only the super-user may set new groups.

## RETURN VALUE

### getgroups

A return value of greater than zero indicates the number of entries placed in the array pointed to by *gidset*. A return value of −1 indicates that an error occurred, and the error code is stored in the global variable **errno**.

### setgroups

A 0 value is returned on success, −1 on error, with a error code stored in **errno**.

## ERRORS

Either call fails if:

| | |
|---|---|
| EFAULT | The address specified for *gidset* is outside the process address space. |

**getgroup** fails if:

| | |
|---|---|
| EINVAL | The argument *gidsetlen* is smaller than the number of groups in the group set. |

**setgroups()** fails if:

| | |
|---|---|
| EPERM | The caller is not the super-user. |

## SEE ALSO

**initgroups**(3)

**NAME**

gethostid – get unique identifier of current host

**SYNOPSIS**

**hostid = gethostid()**
**long hostid;**

**DESCRIPTION**

**gethostid()** returns the 32-bit identifier for the current host, which should be unique across all hosts. On a Sun workstation, this number is taken from the CPU board's ID PROM.

**SEE ALSO**

**hostid**(1)

## NAME

gethostname, sethostname – get/set name of current host

## SYNOPSIS

**int gethostname(name, namelen)**
**char *name;**
**int namelen;**

**int sethostname(name, namelen)**
**char *name;**
**int namelen;**

## DESCRIPTION

**gethostname()** returns the standard host name for the current processor, as previously set by **sethostname**. The parameter *namelen* specifies the size of the array pointed to by *name*. The returned name is null-terminated unless insufficient space is provided.

**sethostname()** sets the name of the host machine to be *name*, which has length *namelen*. This call is restricted to the super-user and is normally used only when the system is bootstrapped.

## RETURN VALUE

If the call succeeds a value of 0 is returned. If the call fails, then a value of −1 is returned and an error code is placed in the global location **errno**.

## ERRORS

The following errors may be returned by these calls:

| | |
|---|---|
| EFAULT | The *name* or *namelen* parameter gave an invalid address. |
| EPERM | The caller was not the super-user. Note that this error only applies to **sethostname**. |

## SEE ALSO

**gethostid**(2)

## BUGS

Host names are limited to **MAXHOSTNAMELEN** (from **<sys/param.h>**) characters, currently 64.

**NAME**

getitimer, setitimer – get/set value of interval timer

**SYNOPSIS**

```
#include <sys/time.h>

int getitimer (which, value)
int which;
struct itimerval *value;

int setitimer (which, value, ovalue)
int which;
struct itimerval *value, *ovalue;
```

**DESCRIPTION**

The system provides each process with three interval timers, defined in **<sys/time.h>**. The **getitimer( )** call stores the current value of the timer specified by **which** into the structure pointed to by *value*. The **setitimer( )** call sets the value of the timer specified by **which** to the value specified in the structure pointed to by *value*, and if *ovalue* is not a NULL pointer, stores the previous value of the timer in the structure pointed to by *ovalue*.

A timer value is defined by the **itimerval** structure, which includes the following members:

```
struct timevalit_interval;/* timer interval */
struct timevalit_value;   /* current value */
```

If **it_value** is non-zero, it indicates the time to the next timer expiration. If **it_interval** is non-zero, it specifies a value to be used in reloading **it_value** when the timer expires. Setting **it_value** to zero disables a timer; however, **it_value** and **it_interval** must still be initialized. Setting **it_interval** to zero causes a timer to be disabled after its next expiration (assuming **it_value** is non-zero).

Time values smaller than the resolution of the system clock are rounded up to this resolution.

The three timers are:

| | |
|---|---|
| **ITIMER_REAL** | Decrements in real time. A **SIGALRM** signal is delivered when this timer expires. |
| **ITIMER_VIRTUAL** | Decrements in process virtual time. It runs only when the process is executing. A **SIGVTALRM** signal is delivered when it expires. |
| **ITIMER_PROF** | Decrements both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the **ITIMER_PROF** timer expires, the **SIGPROF** signal is delivered. Because this signal may interrupt in-progress system calls, programs using this timer must be prepared to restart interrupted system calls. |

**NOTES**

Three macros for manipulating time values are defined in **<sys/time.h>**. **timerclear** sets a time value to zero, **timerisset** tests if a time value is non-zero, and **timercmp** compares two time values (beware that >= and <= do not work with this macro).

**RETURN VALUE**

If the calls succeed, a value of 0 is returned. If an error occurs, the value −1 is returned, and a more precise error code is placed in the global variable **errno**.

**ERRORS**

The possible errors are:

| | |
|---|---|
| EFAULT | The *value* or *ovalue* parameter specified a bad address. |
| EINVAL | The *value* parameter specified a time that was too large to be handled. |

**SEE ALSO**

**sigvec**(2), **gettimeofday**(2)

NAME

getmsg – get next message from a stream

SYNOPSIS

**#include <stropts.h>**

**int getmsg(fd, ctlptr, dataptr, flags)**

**int fd;**

**struct strbuf *ctlptr;**

**struct strbuf *dataptr;**

**int *flags;**

DESCRIPTION

**getmsg()** retrieves the contents of a message (see **intro**(2)) located at the **stream head** read queue from a STREAMS file, and places the contents into user specified buffer(s). The message must contain either a data part, a control part or both. The data and control parts of the message are placed into separate buffers, as described below. The semantics of each part is defined by the STREAMS module that generated the message.

*fd* specifies a file descriptor referencing an open **stream.** *ctlptr* and *dataptr* each point to a **strbuf** structure that contains the following members:

```
int maxlen;/* maximum buffer length */
int len;     /* length of data */
char *buf; /* ptr to buffer */
```

where **buf** points to a buffer in which the data or control information is to be placed, and **maxlen** indicates the maximum number of bytes this buffer can hold. On return, **len** contains the number of bytes of data or control information actually received, or is 0 if there is a zero-length control or data part, or is –1 if no data or control information is present in the message. *flags* may be set to the values 0 or **RS_HIPRI** and is used as described below.

*ctlptr* is used to hold the control part from the message and *dataptr* is used to hold the data part from the message. If *ctlptr* (or *dataptr*) is a NULL pointer or the **maxlen** field is –1, the control (or data) part of the message is not processed and is left on the **stream head** read queue and **len** is set to –1. If the **maxlen** field is set to 0 and there is a zero-length control (or data) part, that zero-length part is removed from the read queue and **len** is set to 0. If the *maxlen* field is set to 0 and there are more than zero bytes of control (or data) information, that information is left on the read queue and **len** is set to 0. If the *maxlen* field in *ctlptr* or *dataptr* is less than, respectively, the control or data part of the message, *maxlen* bytes are retrieved. In this case, the remainder of the message is left on the **stream head** read queue and a non-zero return value is provided, as described below under RETURN VALUE. If information is retrieved from a **priority** message, *flags* is set to **RS_HIPRI** on return.

By default, **getmsg()** processes the first priority or non-priority message available on the **stream head** read queue. However, a process may choose to retrieve only priority messages by setting *flags* to **RS_HIPRI**. In this case, **getmsg()** will only process the next message if it is a priority message.

If **O_NDELAY** has not been set, **getmsg()** blocks until a message, of the type(s) specified by *flags* (priority or either), is available on the **stream head** read queue. If **O_NDELAY** has been set and a message of the specified type(s) is not present on the read queue, **getmsg()** fails and sets **errno** to EAGAIN.

If a hangup occurs on the **stream** from which messages are to be retrieved, **getmsg()** will continue to operate normally, as described above, until the **stream head** read queue is empty. Thereafter, it will return 0 in the **len** fields of *ctlptr* and *dataptr*.

RETURN VALUE

Upon successful completion, a non-negative value is returned. A value of 0 indicates that a full message was read successfully. A return value of **MORECTL** indicates that more control information is waiting for retrieval. A return value of **MOREDATA** indicates that more data is waiting for retrieval. A return value of

MORECTL|MOREDATA indicates that both types of information remain. Subsequent **getmsg()** calls will retrieve the remainder of the message. If an error occurred, a −1 is returned and the global variable **errno** is set to indicate the error.

**ERRORS**

**getmsg()** fails if one or more of the following are true:

| | |
|---|---|
| EAGAIN | The **O_NDELAY** flag is set, and no messages are available. |
| EBADF | *fd* is not a valid file descriptor open for reading. |
| EBADMSG | The queued message to be read is not valid for **getmsg**. |
| EFAULT | *ctlptr*, *dataptr*, or *flags* points to a location outside the allocated address space. |
| EINTR | A signal was caught during the **getmsg()** system call. |
| EINVAL | An illegal value was specified in *flags*, or the **stream** referenced by *fd* is linked under a multiplexor. |
| ENOSTR | A **stream** is not associated with *fd*. |

A **getmsg()** can also fail if a STREAMS error message had been received at the **stream head** before the call to **getmsg**. The error returned is the value contained in the STREAMS error message.

**SEE ALSO**

**intro**(2), **poll**(2), **putmsg**(2), **read**(2), **write**(2)

**NAME**

getpagesize – get system page size

**SYNOPSIS**

**pagesize = getpagesize()**
**int pagesize;**

**DESCRIPTION**

**getpagesize( )** returns the number of bytes in a page. Page granularity is the granularity of many of the memory management calls.

The page size is a *system* page size and may not be the same as the underlying hardware page size.

**SEE ALSO**

**pagesize**(1), **sbrk**(2)

NAME

getpeername – get name of connected peer

SYNOPSIS

**int getpeername(s, name, namelen)**
**int s;**
**struct sockaddr *name;**
**int *namelen;**

DESCRIPTION

**getpeername( )** returns the name of the peer connected to socket *s*. The **int** pointed to by the *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

DIAGNOSTICS

A 0 is returned if the call succeeds, –1 if it fails.

ERRORS

The call succeeds unless:

| | |
|---|---|
| EBADF | The argument *s* is not a valid descriptor. |
| ENOTSOCK | The argument *s* is a file, not a socket. |
| ENOTCONN | The socket is not connected. |
| ENOBUFS | Insufficient resources were available in the system to perform the operation. |
| EFAULT | The *name* parameter points to memory not in a valid part of the process address space. |

SEE ALSO

**accept**(2), **bind**(2), **getsockname**(2), **socket**(2)

**NAME**

getpid, getppid – get process identification

**SYNOPSIS**

```
pid = getpid()
int pid;

ppid = getppid()
int ppid;
```

**DESCRIPTION**

**getpid( )** returns the process ID of the current process. Most often it is used to generate uniquely-named temporary files.

**getppid( )** returns the process ID of the parent of the current process.

**SEE ALSO**

**gethostid**(2)

## NAME

getpriority, setpriority – get/set program scheduling priority

## SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

int getpriority(which, who)
int which, who;

int setpriority(which, who, prio)
int which, who, prio;
```

## DESCRIPTION

The scheduling priority of the process, process group, or user, as indicated by *which* and *who* is obtained with the **getpriority( )** call and set with the **setpriority( )** call. Priorities are values in the range –20 to 20. The default priority is 0; lower priorities cause more favorable scheduling.

*which* is one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**, and *who* is interpreted relative to *which* (a process identifier for **PRIO_PROCESS**, process group identifier for **PRIO_PGRP**, and a user ID for **PRIO_USER**). A zero value of *who* denotes the current process, process group, or user.

The **getpriority( )** call returns the highest priority (lowest numerical value) enjoyed by any of the specified processes. The **setpriority( )** call sets the priorities of all of the specified processes to the value specified by *prio*. If *prio* is less than –20, a value of –20 is used; if it is greater than 20, a value of 20 is used. Only the super-user may lower priorities.

## RETURN VALUE

Since **getpriority( )** can legitimately return the value –1, it is necessary to clear the external variable **errno** prior to the call, then check it afterward to determine if a –1 is an error or a legitimate value. The **setpriority( )** call returns 0 if there is no error, or –1 if there is.

## ERRORS

**getpriority( )** and **setpriority( )** may return one of the following errors:

| | |
|---|---|
| ESRCH | No process was located using the *which* and *who* values specified. |
| EINVAL | *which* was not one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**. |

In addition to the errors indicated above, **setpriority( )** may fail with one of the following errors returned:

| | |
|---|---|
| EPERM | A process was located, but neither its effective nor real user ID matched the effective user ID of the caller, and neither the effective nor the real user ID of the process executing the **setpriority( )** was super-user. |
| EACCES | The call to **getpriority( )** would have changed a process' priority to a value lower than its current value, and the effective user ID of the process executing the call was not that of the super-user. |

## SEE ALSO

**nice**(1), **fork**(2), **renice**(8)

## BUGS

It is not possible for the process executing **setpriority( )** to lower any other process down to its current priority, without requiring super-user privileges.

## NAME

getrlimit, setrlimit – control maximum system resource consumption

## SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

int getrlimit(resource, rlp)
int resource;
struct rlimit *rlp;

int setrlimit(resource, rlp)
int resource;
struct rlimit *rlp;
```

## DESCRIPTION

Limits on the consumption of system resources by the current process and each process it creates may be obtained with the **getrlimit( )** call, and set with the **setrlimit( )** call.

The *resource* parameter is one of the following:

| | |
|---|---|
| **RLIMIT_CPU** | the maximum amount of cpu time (in seconds) to be used by each process. |
| **RLIMIT_FSIZE** | the largest size, in bytes, of any single file that may be created. |
| **RLIMIT_DATA** | the maximum size, in bytes, of the data segment for a process; this defines how far a program may extend its break with the **sbrk( )** (see **brk**(2)) system call. |
| **RLIMIT_STACK** | the maximum size, in bytes, of the stack segment for a process; this defines how far a program's stack segment may be extended automatically by the system. |
| **RLIMIT_CORE** | the largest size, in bytes, of a **core** file that may be created. |
| **RLIMIT_RSS** | the maximum size, in bytes, to which a process's resident set size may grow. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system will prefer to take memory from processes that are exceeding their declared resident set size. |

A resource limit is specified as a soft limit and a hard limit. When a soft limit is exceeded a process may receive a signal (for example, if the cpu time is exceeded), but it will be allowed to continue execution until it reaches the hard limit (or modifies its resource limit). The **rlimit** structure is used to specify the hard and soft limits on a resource,

```
struct rlimit {
        int     rlim_cur;       /* current (soft) limit */
        int     rlim_max;       /* hard limit */
};
```

Only the super-user may raise the maximum limits. Other users may only alter **rlim_cur** within the range from 0 to **rlim_max** or (irreversibly) lower **rlim_max**.

An "infinite" value for a limit is defined as **RLIM_INFINITY (0x7fffffff).**

Because this information is stored in the per-process information, this system call must be executed directly by the shell if it is to affect all future processes created by the shell; **limit** is thus a built-in command to **csh**(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way: a **brk( )** or **sbrk( )** call will fail if the data space limit is reached, or the process will be killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file I/O operation which would create a file that is too large generates a signal **SIGXFSZ**; this normally terminates the process, but may be caught. When the soft CPU time limit is exceeded, a signal **SIGXCPU** is sent to the offending process.

**RETURN VALUE**

A 0 return value indicates that the call succeeded, changing or returning the resource limit. A return value of −1 indicates that an error occurred, and an error code is stored in the global location **errno**.

**ERRORS**

The possible errors are:

| | |
|---|---|
| EFAULT | The address specified by *rlp* is invalid. |
| EINVAL | An invalid *resource* was specified; or in a **setrlimit**( ) call, the new **rlim_cur** exceeds the new **rlim_max**. |
| EPERM | The limit specified to **setrlimit**( ) would have raised the maximum limit value, and the caller is not the super-user. |

**SEE ALSO**

**csh**(1), **sh**(1), **brk**(2) **quotactl**(2),

**BUGS**

There should be **limit** and **unlimit** commands in **sh**(1) as well as in **csh**(1).

NAME

getrusage – get information about resource utilization

SYNOPSIS

```
#include <sys/time.h>
#include <sys/resource.h>

getrusage(who, rusage)
int who;
struct rusage *rusage;
```

DESCRIPTION

**getrusage( )** returns information about the resources utilized by the current process, or all its terminated child processes. The **who** parameter is one of **RUSAGE_SELF** or **RUSAGE_CHILDREN**. The buffer to which *rusage* points will be filled in with the following structure:

```
struct rusage {
        struct timeval ru_utime;        /* user time used */
        struct timeval ru_stime;        /* system time used */
        int     ru_maxrss;
        int     ru_ixrss;               /* integral shared text memory size */
        int     ru_idrss;               /* integral unshared data size */
        int     ru_isrss;                       /* integral unshared stack size */
        int     ru_minflt;              /* page reclaims */
        int     ru_majflt;              /* page faults */
        int     ru_nswap;               /* swaps */
        int     ru_inblock;             /* block input operations */
        int     ru_oublock;             /* block output operations */
        int     ru_msgsnd;              /* messages sent */
        int     ru_msgrcv;              /* messages received */
        int     ru_nsignals;            /* signals received */
        int     ru_nvcsw;               /* voluntary context switches */
        int     ru_nivcsw;              /* involuntary context switches */
};
```

The fields are interpreted as follows:

**ru_utime**
the total amount of time spent executing in user mode. Time is given in seconds:microseconds.

**ru_stime**
the total amount of time spent in the system executing on behalf of the process(es). Time is given in seconds:microseconds.

**ru_maxrss**
the maximum resident set size utilized. Size is given in pages (the size of a page, in bytes, is given by the **getpagesize**(2) system call).

**ru_ixrss**
an "integral" value indicating the amount of memory used by the text segment which was also shared among other processes. This value is expressed in units of pages * clock ticks (1 tick = 1/50 second). The value is calculated by summing the number of shared memory pages in use each time the internal system clock ticks, and then averaging over 1 second intervals.

**ru_idrss**
an integral value of the amount of unshared memory residing in the data segment of a process. The value is given in pages * clock ticks.

**ru_isrss**
an integral value of the amount of unshared memory residing in the stack segment of a process. The value is given in pages * clock ticks.

**ru_minflt**
the number of page faults serviced without any I/O activity; here I/O activity is avoided by "reclaiming" a page frame from the list of pages awaiting reallocation.

**ru_majflt**
the number of page faults serviced which required I/O activity.

**ru_nswap**
the number of times a process was "swapped" out of main memory.

**ru_inblock**
the number of times the file system had to perform input.

**ru_oublock**
the number of times the file system had to perform output.

**ru_msgsnd**
the number of messages sent over sockets.

**ru_msgrcv**
the number of messages received from sockets.

**ru_nsignals**
the number of signals delivered.

**ru_nvcsw**
the number of times a context switch resulted due to a process voluntarily giving up the processor before its time slice was completed (usually to await availability of a resource).

**ru_nivcsw**
the number of times a context switch resulted due to a higher priority process becoming runnable or because the current process exceeded its time slice.

NOTES

The numbers **ru_inblock** and **ru_oublock** account only for real I/O; data supplied by the caching mechanism is charged only to the first process to read or write the data.

ERRORS

**getrusage**( ) will fail if:

| | |
|---|---|
| EINVAL | The **who** parameter is not a valid value. |
| EFAULT | The address specified by the **rusage** argument is not in a valid portion of the process's address space. |

SEE ALSO

**gettimeofday**(2), **wait**(2)

BUGS

There is no way to obtain information about a child process which has not yet terminated.

**NAME**

getsockname – get socket name

**SYNOPSIS**

```
getsockname(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;
```

**DESCRIPTION**

**getsockname( )** returns the current *name* for the specified socket. The *namelen* parameter should be initialized to indicate the amount of space pointed to by *name*. On return it contains the actual size of the name returned (in bytes).

**DIAGNOSTICS**

A 0 is returned if the call succeeds, –1 if it fails.

**ERRORS**

The call succeeds unless:

| | |
|---|---|
| EBADF | The argument *s* is not a valid descriptor. |
| ENOTSOCK | The argument *s* is a file, not a socket. |
| ENOBUFS | Insufficient resources were available in the system to perform the operation. |
| EFAULT | The *name* parameter points to memory not in a valid part of the process address space. |

**SEE ALSO**

**bind**(2), **getpeername**(2), **socket**(2)

**BUGS**

Names bound to sockets in the UNIX domain are inaccessible; **getsockname( )** returns a zero length name.

NAME

getsockopt, setsockopt – get and set options on sockets

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int *optlen;

int setsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int optlen;
```

DESCRIPTION

**getsockopt()** and **setsockopt()** manipulate *options* associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost "socket" level.

When manipulating socket options the level at which the option resides and the name of the option must be specified. To manipulate options at the "socket" level, *level* is specified as **SOL_SOCKET**. To manipulate options at any other level the protocol number of the appropriate protocol controlling the option is supplied. For example, to indicate that an option is to be interpreted by the TCP protocol, *level* should be set to the protocol number of TCP; see **getprotoent**(3N).

The parameters *optval* and *optlen* are used to access option values for **setsockopt**. For **getsockopt()** they identify a buffer in which the value for the requested option(s) are to be returned. For **getsockopt**, *optlen* is a value-result parameter, initially containing the size of the buffer pointed to by *optval*, and modified on return to indicate the actual size of the value returned. If no option value is to be supplied or returned, *optval* may be supplied as 0.

*optname* and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file **<sys/socket.h>** contains definitions for "socket" level options, described below. Options at other protocol levels vary in format and name; consult the appropriate entries in section (4P).

Most socket-level options take an *int* parameter for *optval*. For **setsockopt**, the parameter should non-zero to enable a boolean option, or zero if the option is to be disabled. **SO_LINGER** uses a **struct linger** parameter, defined in **<sys/socket.h>**, which specifies the desired state of the option and the linger interval (see below).

The following options are recognized at the socket level. Except as noted, each may be examined with **getsockopt()** and set with **setsockopt**.

| | |
|---|---|
| **SO_DEBUG** | toggle recording of debugging information |
| **SO_REUSEADDR** | toggle local address reuse |
| **SO_KEEPALIVE** | toggle keep connections alive |
| **SO_DONTROUTE** | toggle routing bypass for outgoing messages |
| **SO_LINGER** | linger on close if data present |
| **SO_BROADCAST** | toggle permission to transmit broadcast messages |
| **SO_OOBINLINE** | toggle reception of out-of-band data in band |
| **SO_SNDBUF** | set buffer size for output |
| **SO_RCVBUF** | set buffer size for input |
| **SO_TYPE** | get the type of the socket (get only) |
| **SO_ERROR** | get and clear error on the socket (get only) |

**SO_DEBUG** enables debugging in the underlying protocol modules. **SO_REUSEADDR** indicates that the rules used in validating addresses supplied in a **bind**(2) call should allow reuse of local addresses. **SO_KEEPALIVE** enables the periodic transmission of messages on a connected socket. Should the

connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified using a SIGPIPE signal. SO_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO_LINGER controls the action taken when unsent messags are queued on socket and a **close**(2) is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system will block the process on the **close**( ) attempt until it is able to transmit the data or until it decides it is unable to deliver the information (a timeout period, termed the linger interval, is specified in the **setsockopt**( ) call when SO_LINGER is requested). If SO_LINGER is disabled and a **close**( ) is issued, the system will process the close in a manner that allows the process to continue as quickly as possible.

The option SO_BROADCAST requests permission to send broadcast datagrams on the socket. Broadcast was a privileged operation in earlier versions of the system. With protocols that support out-of-band data, the SO_OOBINLINE option requests that out-of-band data be placed in the normal data input queue as received; it will then be accessible with **recv**( ) or **read**( ) calls without the MSG_OOB flag. SO_SNDBUF and SO_RCVBUF are options to adjust the normal buffer sizes allocated for output and input buffers, respectively. The buffer size may be increased for high-volume connections, or may be decreased to limit the possible backlog of incoming data. The system places an absolute limit on these values. Finally, SO_TYPE and SO_ERROR are options used only with **getsockopt**. SO_TYPE returns the type of the socket, such as SOCK_STREAM; it is useful for servers that inherit sockets on startup. SO_ERROR returns any pending error on the socket and clears the error status. It may be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors.

## RETURN VALUE

A 0 is returned if the call succeeds, −1 if it fails.

## ERRORS

The call succeeds unless:

| | |
|---|---|
| EBADF | The argument *s* is not a valid descriptor. |
| ENOTSOCK | The argument *s* is a file, not a socket. |
| ENOPROTOOPT | The option is unknown at the level indicated. |
| EFAULT | The address pointed to by *optval* is not in a valid part of the process address space. For **getsockopt**, this error may also be returned if *optlen* is not in a valid part of the process address space. |

## SEE ALSO

**ioctl**(2), **socket**(2), **getprotoent**(3N)

## BUGS

Several of the socket options should be handled at lower levels of the system.

## NAME

gettimeofday, settimeofday – get or set the date and time

## SYNOPSIS

```
#include <sys/time.h>

int gettimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;

int settimeofday(tp, tzp)
struct timeval *tp;
struct timezone *tzp;
```

## DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the **gettimeofday( )** call, and set with the **settimeofday( )** call. The current time is expressed in elapsed seconds and microseconds since 00:00 GMT, January 1, 1970 (zero hour). The resolution of the system clock is hardware dependent; the time may be updated continuously, or in "ticks."

*tp* points to a **timeval** structure, which includes the following members:

```
long tv_sec;  /* seconds since Jan. 1, 1970 */
long tv_usec; /* and microseconds */
```

If *tp* is a NULL pointer, the current time information is not returned or set.

*tzp* points to a **timezone( )** structure, which includes the following members:

```
int tz_minuteswest; /* of Greenwich */
int tz_dsttime;     /* type of dst correction to apply */
```

The **timezone( )** structure indicates the local time zone (measured in minutes westward from Greenwich), and a flag that indicates the type of Daylight Saving Time correction to apply. Note: this flag does *not* indicate whether Daylight Saving Time is currently in effect.

Also note that the offset of the local time zone from GMT may change over time, as may the rules for Daylight Saving Time correction. The **localtime( )** routine (see **ctime**(3)) obtains this information from a file rather than from **gettimeofday**. Programs should use **localtime( )** to convert dates and times; the **timezone( )** structure is filled in by **gettimeofday( )** for backward compatibility with existing programs.

The flag indicating the type of Daylight Saving Time correction should have one of the following values (as defined in **<sys/time.h>**):

| | |
|---|---|
| 0 | **DST_NONE**: Daylight Savings Time not observed |
| 1 | **DST_USA**: United States DST |
| 2 | **DST_AUST**: Australian DST |
| 3 | **DST_WET**: Western European DST |
| 4 | **DST_MET**: Middle European DST |
| 5 | **DST_EET**: Eastern European DST |
| 6 | **DST_CAN**: Canadian DST |
| 7 | **DST_GB**: Great Britain and Eire DST |
| 8 | **DST_RUM**: Rumanian DST |
| 9 | **DST_TUR**: Turkish DST |
| 10 | **DST_AUSTALT**: Australian-style DST with shift in 1986 |

If *tzp* is a NULL pointer, the time zone information is not returned or set.

Only the super-user may set the time of day or the time zone.

## RETURN

A –1 return value indicates an error occurred; in this case an error code is stored in the global variable **errno**.

**ERRORS**

The following error codes may be set in **errno**:

| | |
|---|---|
| EFAULT | An argument address referenced invalid memory. |
| EPERM | A user other than the super-user attempted to set the time or time zone. |

**SEE ALSO**

**date**(1V), **adjtime**(2), **ctime**(3)

**BUGS**

Time is never correct enough to believe the microsecond values. There should a mechanism by which, at least, local clusters of systems might synchronize their clocks to millisecond granularity.

**NAME**

getuid, geteuid – get user identity

**SYNOPSIS**

**uid = getuid()**
**int uid;**

**euid = geteuid()**
**int euid;**

**DESCRIPTION**

**getuid( )** returns the real user ID of the current process, **geteuid( )** the effective user ID.

The real user ID identifies the person who is logged in. The effective user ID gives the process additional permissions during execution of "set-user-ID" mode processes, which use **getuid( )** to determine the real-user-id of the process that invoked them.

**SEE ALSO**

**getgid**(2), **setreuid**(2)

**NAME**

ioctl – control device

**SYNOPSIS**

**int ioctl(des, request, arg)**
**int des, request;**

**DESCRIPTION**

**ioctl()** performs a special function on the object referred to by the open descriptor *des*. The set of functions that may be performed depends on the object that *des* refers to. For example, many operating characteristics of character special files (for instance, terminals) may be controlled with **ioctl()** requests. The writeups in section 4 discuss how **ioctl()** applies to various objects.

The *request* codes for particular functions are specified in include files specific to objects or to families of objects; the writeups in section 4 indicate which include files specify which *request*s.

For most **ioctl()** functions, *arg* is a pointer to data to be used by the function or to be filled in by the function. Other functions may ignore *arg* or may treat it directly as a data item; they may, for example, be passed an **int** value.

**RETURN VALUE**

If an error has occurred, a value of −1 is returned and **errno** is set to indicate the error.

If no error has occurred, a value of 0 is returned by most functions. Some specialized functions may return non-zero values on success; see the description of the function in the writeup for the object.

**ERRORS**

**ioctl()** will fail if one or more of the following are true:

| | |
|---|---|
| EBADF | *des* is not a valid descriptor. |
| ENOTTY | The specified request does not apply to the kind of object to which the descriptor *des* refers. |
| EINVAL | *request* or *arg* is not valid. |
| EFAULT | *request* requires a data transfer to or from a buffer pointed to by *arg*, but some part of the buffer is outside the process's allocated space. |

**ioctl()** will also fail if the object on which the function is being performed detects an error. In this case, an error code specific to the object and the function will be returned.

**SEE ALSO**

**execve**(2), **fcntl**(2V), **filio**(4), **mtio**(4), **sockio**(4), **streamio**(4), **termio**(4)

## NAME

kill – send a signal to a process or a group of processes

## SYNOPSIS

**kill(pid, sig)**

**int pid, sig;**

## DESCRIPTION

**kill()** sends the signal *sig* to a process or a group of processes. The process or group of processes to which the signal is to be sent is specified by *pid*. *sig* may be one of the signals specified in **sigvec**(2), or it may be 0, in which case error checking is performed but no signal is actually sent. This can be used to check the validity of *pid*.

The real or effective user ID of the sending process must match the real or saved set-user ID of the receiving process, unless the effective user ID of the sending process is super-user. A single exception is the signal SIGCONT, which may always be sent to any descendant of the current process.

In the following discussion, "system processes" are processes, such as processes 0 and 2, that are not running a regular user program.

If *pid* is greater than zero, the signal is sent to the process whose process ID is equal to *pid*. *pid* may equal 1.

If *pid* is 0, the signal is sent to all processes, except system processes, process 1, and the process sending the signal, whose process group ID is equal to the process group ID of the sender; this is a variant of **killpg**(2).

If *pid* is −1 and the effective user ID of the sender is not super-user, the signal is sent to all processes, except system processes, process 1, and the process sending the signal, whose real or saved set-user ID matches the real or effective ID of the sender.

If *pid* is −1 and the effective user ID of the sender is super-user, the signal is sent to all processes except system processes, process 1, and the process sending the signal.

If *pid* is negative but not −1, the signal is sent to all processes, except system processes, process 1, and the process sending the signal, whose process group ID is equal to the absolute value of *pid*; this is a variant of **killpg**(2).

Processes may send signals to themselves.

## SYSTEM V DESCRIPTION

If a signal is sent to a group of processes (as with, if *pid* is 0 or negative), and if the process sending the signal is a member of that group, the signal is sent to that process as well.

The signal SIGKILL cannot be sent to process 1.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**kill()** will fail and no signal will be sent if any of the following occur:

| | |
|---|---|
| EINVAL | *sig* is not a valid signal number. |
| ESRCH | No process can be found corresponding to that specified by *pid*. |
| EPERM | The effective user ID of the sending process is not super-user, and neither its real nor effective user ID matches the real or saved set-user ID of the receiving process. |

## SYSTEM V ERRORS

**kill()** will also fail, and no signal will be sent, if the following occurs:

| | |
|---|---|
| EINVAL | *sig* is SIGKILL and *pid* is 1. |

**SEE ALSO**

**getpid**(2), **killpg**(2), **setpgrp**(2V), **sigvec**(2)

NAME

killpg – send signal to a process group

SYNOPSIS

**int killpg(pgrp, sig)**

int pgrp, sig;

DESCRIPTION

**killpg**( ) sends the signal *sig* to the process group *pgrp*. See **sigvec**(2) for a list of signals.

The real or effective user ID of the sending process must match the real or saved set-user ID of the receiving process, unless the effective user ID of the sending process is super-user. A single exception is the signal **SIGCONT**, which may always be sent to any descendant of the current process.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and the global variable **errno** is set to indicate the error.

ERRORS

**killpg**( ) will fail and no signal will be sent if any of the following occur:

| | |
|---|---|
| EINVAL | *sig* is not a valid signal number. |
| ESRCH | No processes were found in the specified process group. |
| EPERM | The effective user ID of the sending process is not super-user, and neither its real nor effective user ID matches the real or saved set-user ID of one or more of the target processes. |

SEE ALSO

**kill**(2V), **setpgrp**(2V), **sigvec**(2)

## NAME

link – make a hard link to a file

## SYNOPSIS

**int link(name1, name2)**
**char *name1, *name2;**

## DESCRIPTION

*name1* points to a path name naming an existing file. *name2* points to a path name naming a new directory entry to be created. A hard link to the first file is created; the link has the name pointed to by *name2*. The file named by *name1* must exist.

With hard links, both files must be on the same file system. Unless the caller is the super-user, the file named by *name1* must not be a directory. Both the old and the new **link**() share equal access and rights to the underlying object.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**link**() will fail and no link will be created if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *name1* or *name2* is not a directory. |
| ENAMETOOLONG | The length of a component of *name1* or *name2* exceeds 255 characters, or the length of *name1* or *name2* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of *name1* or *name2* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *name1* or *name2*. |
| EACCES | The requested link requires writing in a directory for which write permission is denied. |
| ELOOP | Too many symbolic links were encountered in translating *name1* or *name2*. |
| ENOENT | The file referred to by *name1* does not exist. |
| EEXIST | The link referred to by *name2* does exist. |
| EPERM | The file named by *name1* is a directory and the effective user ID is not super-user. |
| EXDEV | The link named by *name2* and the file named by *name1* are on different file systems. |
| ENOSPC | The directory in which the entry for the new link is being placed cannot be extended because there is no space left on the file system containing the directory. |
| EDQUOT | The directory in which the entry for the new link is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted. |
| EIO | An I/O error occurred while reading from or writing to the file system to make the directory entry. |
| EROFS | The requested link requires writing in a directory on a read-only file system. |
| EFAULT | One of the path names specified is outside the process's allocated address space. |

## SEE ALSO

**symlink**(2), **unlink**(2)

**NAME**

listen – listen for connections on a socket

**SYNOPSIS**

**listen(s, backlog)**
**int s, backlog;**

**DESCRIPTION**

To accept connections, a socket is first created with **socket**(2), a backlog for incoming connections is specified with **listen**( ) and then the connections are accepted with **accept**(2). The **listen**( ) call applies only to sockets of type **SOCK_STREAM** or **SOCK_SEQPACKET**.

The *backlog* parameter defines the maximum length the queue of pending connections may grow to. If a connection request arrives with the queue full the client will receive an error with an indication of ECONNREFUSED.

**RETURN VALUE**

A 0 return value indicates success; –1 indicates an error.

**ERRORS**

The call fails if:

| | |
|---|---|
| EBADF | The argument *s* is not a valid descriptor. |
| ENOTSOCK | The argument *s* is not a socket. |
| EOPNOTSUPP | The socket is not of a type that supports the operation **listen**. |

**SEE ALSO**

**accept**(2), **connect**(2), **socket**(2)

**BUGS**

The *backlog* is currently limited (silently) to 5.

**NAME**

lseek, tell – move read/write pointer

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/file.h>

off_t lseek(des, offset, whence)
int des;
off_t offset;
int whence;
```

**DESCRIPTION**

The descriptor *des* refers to a file or device open for reading and/or writing. **lseek()** sets the file pointer associated with *des* as follows:

If *whence* is **L_SET**, the pointer is set to *offset* bytes.

If *whence* is **L_INCR**, the pointer is set to its current location plus *offset*.

If *whence* is **L_XTND**, the pointer is set to the size of the file plus *offset*.

Some devices are incapable of seeking. The value of the pointer associated with such a device is undefined.

The obsolete function **tell(fildes)** is identical to **lseek(fildes, 0L, L_INCR)**.

**NOTES**

Seeking far beyond the end of a file, then writing, may create a gap or "hole", which occupies no physical space and reads as zeros.

**RETURN VALUE**

Upon successful completion, the resulting pointer location, as measured in bytes from beginning of the file, is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

**ERRORS**

**lseek()** will fail and the file pointer will remain unchanged if:

| | |
|---|---|
| EBADF | *des* is not an open file descriptor. |
| ESPIPE | *des* is associated with a pipe or a socket. |
| EINVAL | *whence* is not a proper value. |

**SEE ALSO**

**dup**(2), **open**(2V)

## NAME

mincore – determine residency of memory pages

## SYNOPSIS

**mincore(addr, len, vec)**
**caddr_t addr; int len; result char *vec;**

## DESCRIPTION

**mincore**( ) returns the primary memory residency status of pages in the address space covered by mappings in the range [*addr*, *addr* + *len*). The status is returned as a char-per-page in the character array referenced by **vec* (which the system assumes to be large enough to encompass all the pages in the address range). The least significant bit of each character is set to 1 to indicate that the referenced page is in primary memory, 0 if it is not. The settings of other bits in each character is undefined and may contain other information in the future.

## RETURN VALUE

**mincore**( ) returns 0 on success, -1 on failure.

## ERRORS

**mincore**( ) will fail if:

| | |
|---|---|
| EFAULT | **vec* includes an out-of-range or otherwise inaccessible address. |
| EINVAL | *addr* is not a multiple of the page size as returned by **getpagesize**(2). |
| ENOMEM | Addresses in the range [*addr*, *addr* + *len*) are invalid for the address space of a process, or specify one or more pages which are not mapped. |

## SEE ALSO

**mmap**(2)

NAME

mkdir – make a directory file

SYNOPSIS

**int mkdir(path, mode)**
**char *path;**
**int mode;**

DESCRIPTION

**mkdir()** creates a new directory file with name *path*. The mode of the new file is initialized from *mode*.

The low-order 9 bits of *mode* are modified such that all bits set in the process's file mode creation mask are cleared (see **umask**(2)).

The set-gid bit of *mode* is ignored. The set-gid bit of the new file is inherited from that of the parent directory.

The directory's owner ID is set to the process's effective user ID.

The directory's group ID is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2)) and the set-gid bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

RETURN VALUE

A 0 return value indicates success. A −1 return value indicates an error, and an error code is stored in **errno**.

ERRORS

**mkdir()** will fail and no directory will be created if:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EROFS | The file referred to by *path* resides on a read-only file system. |
| EEXIST | The file referred to by *path* exists. |
| ENOSPC | The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory. |
| ENOSPC | The new directory cannot be created because there is no space left on the file system which will contain the directory. |
| ENOSPC | There are no free inodes on the file system on which the file is being created. |
| EDQUOT | The directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted. |
| EDQUOT | The new directory cannot be created because the user's quota of disk blocks on the file system which will contain the directory has been exhausted. |
| EDQUOT | The user's quota of inodes on the file system on which the file is being created has been exhausted. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EFAULT | *path* points outside the process's allocated address space. |

**SEE ALSO**

**chmod**(2), **mount**(2), **rmdir**(2), **stat**(2), **umask**(2)

NAME

mknod – make a special file

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int mknod(path, mode, dev)
char *path;
int mode, dev;
```

DESCRIPTION

**mknod()** creates a new file named by the path name pointed to by *path*. The mode of the new file (including file type bits) is initialized from *mode*. The values of the file type bits which are permitted are:

```
#define  S_IFCHR    0020000    /* character special */
#define  S_IFBLK    0060000    /* block special */
#define  S_IFREG    0100000    /* regular */
#define  S_IFIFO    0010000    /* FIFO special */
```

Values of *mode* other than those above are undefined and should not be used.

The protection part of the mode is modified by the process's mode mask (see **umask**(2)).

The owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2)) and the set-gid bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

If mode indicates a block or character special file, *dev* is a configuration dependent specification of a character or block I/O device. If *mode* does not indicate a block special or character special device, *dev* is ignored.

**mknod()** may be invoked only by the super-user for file types other than FIFO special.

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

ERRORS

**mknod()** fails and the file mode remains unchanged if:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EPERM | An attempt was made to create a file of type other than FIFO special and the process's effective user ID is not super-user. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EISDIR | The specified *mode* would have created a directory. |
| ENOSPC | The directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory. |

| | |
|---|---|
| ENOSPC | There are no free inodes on the file system on which the file is being created. |
| EDQUOT | The directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted. |
| EDQUOT | The user's quota of inodes on the file system on which the node is being created has been exhausted. |
| EROFS | The file referred to by *path* resides on a read-only file system. |
| EEXIST | The file referred to by *path* exists. |
| EFAULT | *path* points outside the process's allocated address space. |

**SEE ALSO**

**chmod**(2), **stat**(2), **umask**(2)

NAME

mmap – map pages of memory

SYNOPSIS

```
#include <sys/types.h>
#include <sys/mman.h>
caddr_t mmap(addr, len, prot, flags, fd, off)
caddr_t addr;
int len, prot, flags, fd;
off_t off;
```

DESCRIPTION

**mmap()** establishes a mapping between the process's address space at an address *paddr* for *len* bytes to the memory object represented by *fd* at *off* for *len* bytes. The value of *paddr* is an implementation-dependent function of the parameter *addr* and values of *flags*, further described below. A successful **mmap()** call returns *paddr* as its result. The address ranges covered by [*paddr, paddr + len*) and [*off, off + len*) must be legitimate for the address space of a process and the object in question, respectively.

The mapping established by **mmap()** replaces any previous mappings for the process's pages in the range [*paddr, paddr + len*).

The parameter *prot* determines whether read, execute, write, or some combination of accesses are permitted to the pages being mapped. The protection options are defined in **<sys/mman.h>** as:

```
#define PROT_READ       0x4     /* page can be read */
#define PROT_WRITE      0x2     /* page can be written */
#define PROT_EXECUTE    0x1     /* page can be executed */
#define PROT_NONE       0x0     /* page can not be accessed */
```

Not all implementations literally provide all possible combinations. **PROT_WRITE** is often implemented as **PROT_READ|PROT_WRITE** and **PROT_EXECUTE** as **PROT_READ|PROT_EXECUTE.** However, no implementation will permit a write to succeed where **PROT_WRITE** has not been set. The behavior of **PROT_WRITE** can be influenced by setting **MAP_PRIVATE** in the *flags* parameter, described below.

The parameter *flags* provides other information about the handling of the mapped pages. The options are defined in **<sys/mman.h>** as:

```
#define MAP_SHARED      1       /* Share changes */
#define MAP_PRIVATE     2       /* Changes are private */
#define MAP_TYPE        0xf     /* Mask for type of mapping */
#define MAP_FIXED       0x10    /* Interpret addr exactly */
#define MAP_RENAME      0x20    /* Assign page to file */
```

**MAP_SHARED** and **MAP_PRIVATE** describe the disposition of write references to the memory object. If **MAP_SHARED** is specified, write references will change the memory object. If **MAP_PRIVATE** is specified, the initial write reference will create a private copy of the memory object page and redirect the mapping to the copy. The mapping type is retained across a **fork**(2).

**MAP_FIXED** informs the system that the value of *paddr* must be *addr*, exactly. The use of **MAP_FIXED** is discouraged, as it may prevent an implementation from making the most effective use of system resources.

When **MAP_FIXED** is not set, the system uses *addr* as a hint in an implementation-defined manner to arrive at *paddr*. The *paddr* so chosen will be an area of the address space which the system deems suitable for a mapping of *len* bytes to the specified object. All implementations interpret an *addr* value of zero as granting the system complete freedom in selecting *paddr*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the system selects a value for *paddr*, it will never place a mapping at address 0, nor will it replace any extant mapping, nor map into areas considered part of the potential data or stack "segments".

**MAP_RENAME** causes the pages currently mapped in the range [*paddr, paddr + len*) to be effectively renamed to be the file pages at [*off, off + len*). The currently mapped pages must be mapped as **MAP_PRIVATE**. **MAP_RENAME** implies a **MAP_FIXED** interpretation of *addr*. *fd* must be open for writing. **MAP_RENAME** affects the size of the memory object referenced by *fd*: the size is max(*off + len - 1, flen*) (where *flen* was the previous length of the object). After the pages are renamed, a mapping to them is reestablished with the parameters as specified in the renaming **mmap**.

The parameter *off* is constrained to be aligned and sized according to the value returned by **getpagesize** (2). When **MAP_FIXED** is specified, the parameter *addr* must also meet these constraints. The system performs mapping operations over whole pages. Thus, while the parameter *len* need not meet a size or alignment constraint, the system will include in any mapping operation any partial page specified by the range [*paddr, paddr + len*).

It should be noted that the system will always zero-fill any partial pages at the end of an object. Further, the system will never write out any modified portions of the last page of an object which are beyond its end. References to whole pages following the end of an object will result in the delivery of a **SIGBUS** signal. **SIGBUS** signals may also be delivered on various filesystem conditions, including quota exceeded errors.

## RETURN VALUE

A successful **mmap()** returns the address at which the mapping was placed (*paddr*). A failing **mmap()** returns −1.

## ERRORS

**mmap()** will fail if:

| | |
|---|---|
| EBADF | *fd* is not open. |
| EACCES | *fd* is not open for read and **PROT_READ** or **PROT_EXECUTE** were specified, or *fd* is not open for write and **PROT_WRITE** was specified for a **MAP_SHARED** type mapping. |
| ENXIO | Addresses in the range [*off, off + len*) are invalid for *fd*. |
| EINVAL | The arguments *addr* (if **MAP_FIXED** was specified) and *off* are not multiples of the page size as returned by **getpagesize** (2). |
| EINVAL | The **MAP_TYPE** field in *flags* is invalid (neither **MAP_PRIVATE** or **MAP_SHARED**). |
| ENODEV | *fd* refers to an object for which **mmap()** is meaningless, such as a terminal; or if the object does support **mmap()** and **MAP_RENAME** was specified then the object is unable to support **MAP_RENAME** (for instance, **MAP_RENAME** to a frame buffer or other device). |
| ENOMEM | **MAP_FIXED** was specified, and the range [*addr, addr + len*) exceeds that allowed for the address space of a process; OR **MAP_FIXED** was not specified and there is insufficient room in the address space to effect the mapping. |
| ENOSPC | **MAP_RENAME** was specified and there is no space left on the filesystem to hold the pages. |
| ETXTBSY | One or more pages specified by an **mmap() MAP_RENAME** operation are not mapped **MAP_PRIVATE**. |

## SEE ALSO

**fork**(2), **getpagesize**(2), **munmap**(2), **mprotect**(2)

## BUGS

**MAP_RENAME** is not implemented.

**NAME**

mount – mount file system

**SYNOPSIS**

```
#include <sys/mount.h>

int mount(type, dir, M_NEWTYPE|flags, data)
char *type;
char *dir;
int flags;
caddr_t data;
```

**DESCRIPTION**

**mount()** attaches a file system to a directory. After a successful return, references to directory *dir* will refer to the root directory on the newly mounted file system. *dir* is a pointer to a null-terminated string containing a path name. *dir* must exist already, and must be a directory. Its old contents are inaccessible while the file system is mounted.

**mount()** may be invoked only by the super-user.

The *flags* argument is constructed by the logical OR of the following bits (defined in **<sys/mount.h>**):

**M_RDONLY**
mount filesystem read-only.

**M_NOSUID**
ignore set-uid bit on execution.

**M_NEWTYPE**
this flag must always be set.

**M_GRPID**
use BSD file-creation semantics (see **open**(2V)).

**M_REMOUNT**
change options on an existing mount.

**M_NOSUB**
disallow mounts beneath this filesystem.

**M_MULTI**
evaluate each pathname as a unit, rather than one component at a time (This option has no effect unless the particular filesystem type is specified as supporting it.)

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

The *type* string indicates the type of the filesystem. *data* is a pointer to a structure which contains the type specific arguments to mount. Below is a list of the filesystem types supported and the type specific arguments to each:

"4.2" 4.3

```
struct ufs_args {
        char    *fspec;     /* Block special file to mount */
};
```

"nfs"

```
#include        <nfs/nfs.h>
#include        <netinet/in.h>
struct nfs_args {
        struct sockaddr_in  *addr; /* file server address */
        fhandle_t  *fh;         /* File handle to be mounted */
        int        flags;       /* flags */
```

```
        int     wsize;      /* write size in bytes */
        int     rsize;      /* read size in bytes */
        int     timeo;      /* initial timeout in .1 secs */
        int     retrans;    /* times to retry send */
        char    *hostname;  /* server's hostname */
        int     acregmin;   /* attr cache file min secs */
        int     acregmax;   /* attr cache file max secs */
        int     acdirmin;   /* attr cache dir min secs */
        int     acdirmax;   /* attr cache dir max secs */
        char    *netname;   /* server's netname */
};
```

**RETURN VALUE**

Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

**ERRORS**

**mount()** fails when one of the following occurs:

| | |
|---|---|
| EPERM | The caller is not the super-user. |
| ENODEV | The file system type specified by *type* is not valid or is not configured into the system. |
| ENAMETOOLONG | The length of a component of the path name of *dir* exceeds 255 characters, or the length of the entire path name of *dir* exceeds 1023 characters. |
| ENOENT | A component of *dir* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *dir*. |
| ENOTDIR | The file named by *dir* is not a directory. |
| EBUSY | Another process currently holds a reference to *dir*. |
| EFAULT | *dir* points outside the process's allocated address space. |
| ELOOP | Too many symbolic links were encountered in translating the path name of *dir*. |

For a 4.2 file system, **mount()** fails when one of the following occurs:

| | |
|---|---|
| ENOTBLK | *fspec* is not a block device. |
| ENXIO | The major device number of *fspec* is out of range (this indicates no device driver exists for the associated hardware). |
| EMFILE | No space remains in the mount table. |
| EINVAL | The super block for the file system had a bad magic number or an out of range block size. |
| ENOMEM | Not enough memory was available to read the cylinder group information for the file system. |
| ENOTDIR | A component of the path prefix of *fspec* is not a directory. |
| ENAMETOOLONG | The length of a component of the path name of *fspec* exceeds 255 characters, or the length of the entire path name of *fspec* exceeds 1023 characters. |
| ENOENT | A component of *fspec* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *fspec*. |
| EFAULT | *fspec* points outside the process's allocated address space. |
| ELOOP | Too many symbolic links were encountered in translating the path name of *fspec*. |

EIO An I/O error occurred while reading from or writing to the file system.

**SEE ALSO**

**unmount**(2), **open**(2V), **mount**(8)

**BUGS**

Some of the error codes need translation to more obvious messages.

**NAME**

mprotect – set protection of memory mapping

**SYNOPSIS**

**#include <sys/mman.h>**
**mprotect(addr, len, prot)**
**caddr_t addr; int len, prot;**

**DESCRIPTION**

**mprotect( )** changes the access protections on the mappings specified by the range [*addr*, *addr* + *len*) to be that specified by *prot*. Legitimate values for *prot* are the same as those permitted for **mmap**(2).

**RETURN VALUE**

**mprotect( )** returns 0 on success, −1 on failure.

**ERRORS**

**mprotect( )** will fail if:

| | |
|---|---|
| EACCES | *prot* specifies a protection which violates the access permission the process has to the underlying memory object. |
| EINVAL | *addr* is not a multiple of the page size as returned by **getpagesize**(2). |
| ENOMEM | Addresses in the range [*addr*, *addr* + *len*) are invalid for the address space of a process, or specify one or more pages which are not mapped. |

When **mprotect( )** fails for reasons other than EINVAL, the protections on some of the pages in the range [*addr*, *addr* + *len*) will have been changed. If the error occurs on some page at address *addr2*, then the protections of all whole pages in the range [*addr*, *addr2*) have been modified.

**SEE ALSO**

**getpagesize**(2), **mmap**(2)

## NAME

msgctl – message control operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;
```

## DESCRIPTION

**msgctl()** provides a variety of message control operations as specified by *cmd*. The following *cmd*s are available:

**IPC_STAT** Place the current value of each member of the data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in **intro**(2). **{READ}**

**IPC_SET** Set the value of the following members of the data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:

```
msg_perm.uid
msg_perm.gid
msg_perm.mode /* only low 9 bits */
msg_qbytes
```

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid*. Only super-user can raise the value of **msg_qbytes**.

**IPC_RMID** Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and data structure associated with it. This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid*.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**msgctl()** will fail if:

EINVAL *msqid* is not a valid message queue identifier.

EINVAL *cmd* is not a valid command.

EACCES *cmd* is equal to IPC_STAT and **{READ}** operation permission is denied to the calling process (see **intro**(2)).

EPERM *cmd* is equal to IPC_RMID or IPC_SET. The effective user ID of the calling process is not equal to that of super-user, or to the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid*.

EPERM *cmd* is equal to IPC_SET, an attempt is being made to increase to the value of **msg_qbytes**, and the effective user ID of the calling process is not equal to that of super-user.

EFAULT *buf* points to an illegal address.

**SEE ALSO**

**intro**(2), **msgget**(2), **msgop**(2)

## NAME

msgget – get message queue

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key, msgflg)
key_t key;
int msgflg;
```

## DESCRIPTION

**msgget()** returns the message queue identifier associated with **key**.

A message queue identifier and associated message queue and data structure (see **intro**(2)) are created for **key()** if one of the following are true:

- *key* is equal to **IPC_PRIVATE**.
- *key* does not already have a message queue identifier associated with it, and (*msgflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new message queue identifier is initialized as follows:

- **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **msg_perm.mode** are set equal to the low-order 9 bits of *msgflg*.
- **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** are set equal to 0.
- **msg_ctime** is set equal to the current time.
- **msg_qbytes** is set equal to the system-wide standard value of the maximum number of bytes allowed on a message queue.

## RETURN VALUE

Upon successful completion, a non-negative integer, namely a message queue identifier, is returned. Otherwise, a value of –1 is returned and **errno** is set to indicate the error.

## ERRORS

**msgget()** will fail if one or more of the following are true:

| | |
|---|---|
| EACCES | A message queue identifier exists for **key**, but operation permission (see **intro**(2)) as specified by the low-order 9 bits of **msgflg** would not be granted. |
| ENOENT | A message queue identifier does not exist for **key()** and (**msgflg** & **IPC_CREAT**) is "false". |
| ENOSPC | A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded. |
| EEXIST | A message queue identifier exists for **key()** but ( (**msgflg** & **IPC_CREAT**) & (**msgflg** & **IPC_EXCL**)) is "true". |

## SEE ALSO

**intro**(2), **msgctl**(2), **msgop**(2)

NAME

msgop, msgsnd, msgrcv – message operations

SYNOPSIS

```
#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

int msgsnd(msqid, msgp, msgsz, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz, msgflg;

int msgrcv(msqid, msgp, msgsz, msgtyp, msgflg)
int msqid;
struct msgbuf *msgp;
int msgsz;
long msgtyp;
int msgflg;
```

DESCRIPTION

**msgsnd( )** is used to send a message to the queue associated with the message queue identifier specified by *msqid*. {WRITE} *msgp* points to a structure containing the message. This structure is composed of the following members:

```
long    mtype;      /* message type */
char    mtext[];    /* message text */
```

*mtype* is a positive integer that can be used by the receiving process for message selection (see **msgrcv** below). *mtext* is any text of length *msgsz* bytes. *msgsz* can range from 0 to a system-imposed maximum.

*msgflg* specifies the action to be taken if one or more of the following are true:

The number of bytes already on the queue is equal to *msg_qbytes* (see **intro** (2)).

The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

If **(msgflg & IPC_NOWAIT)** is return immediately.

If **(msgflg & IPC_NOWAIT)** is the calling process will suspend execution until one of the following occurs:

The condition responsible for the suspension no longer exists, in which case the message is sent.

*msqid* is removed from the system (see **msgctl**(2)). When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.

The calling process receives a signal that is to be caught. In this case the message is not sent and the calling process resumes execution in the manner prescribed in **signal**(3).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see **intro**(2)).

*msg_qnum* is incremented by 1.

*msg_lspid* is set equal to the process ID of the calling process.

*msg_stime* is set equal to the current time.

**msgrcv( )** reads a message from the queue associated with the message queue identifier specified by *msqid* and places it in the structure pointed to by *msgp*. {READ} This structure is composed of the following members:

```
long    mtype;      /* message type */
char    mtext[];    /* message text */
```

*mtype* is the received message's type as specified by the sending process. *mtext* is the text of the message. *msgsz* specifies the size in bytes of *mtext*. The received message is truncated to *msgsz* bytes if it is larger than *msgsz* and (**msgflg** & MSG_NOERROR) is The truncated part of the message is lost and no indication of the truncation is given to the calling process.

*msgtyp* specifies the type of message requested as follows:

If *msgtyp* is equal to 0, the first message on the queue is received.

If *msgtyp* is greater than 0, the first message of type *msgtyp* is received.

If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* is received.

*msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

If (**msgflg** & IPC_NOWAIT) is of −1 and **errno** set to ENOMSG.

If (**msgflg** & IPC_NOWAIT) is following occurs:

A message of the desired type is placed on the queue.

*msqid* is removed from the system. When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.

The calling process receives a signal that is to be caught. In this case a message is not received and the calling process resumes execution in the manner prescribed in **signal**(3).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see intro (2)).

*msg_qnum* is decremented by 1.

*msg_lrpid* is set equal to the process ID of the calling process.

*msg_rtime* is set equal to the current time.

## RETURN VALUES

Upon successful completion, the return value is as follows:

**msgsnd**( ) returns a value of 0.

**msgrcv**( ) returns a value equal to the number of bytes actually placed into *mtext*.

Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**msgsnd**( ) will fail and no message will be sent if one or more of the following are true:

| | |
|---|---|
| EINVAL | *msqid* is not a valid message queue identifier. |
| EIDRM | The message queue referred to by *msqid* was removed from the system. |
| EACCES | Operation permission is denied to the calling process (see **intro**(2)). |
| EINVAL | *mtype* is less than 1. |
| EAGAIN | The message cannot be sent for one of the reasons cited above and (**msgflg** & IPC_NOWAIT) is |
| EINVAL | *msgsz* is less than zero or greater than the system-imposed limit. |
| EFAULT | *msgp* points to an illegal address. |
| EINTR | The call was interrupted by the delivery of a signal. |

**msgrcv**() will fail and no message will be received if one or more of the following are true:

| | |
|---|---|
| EINVAL | *msqid* is not a valid message queue identifier. |
| EIDRM | The message queue referred to by *msqid* was removed from the system. |
| EACCES | Operation permission is denied to the calling process. |
| EINVAL | *msgsz* is less than 0. |
| E2BIG | *mtext* is greater than *msgsz* and (**msgflg** & **MSG_NOERROR**) is |
| ENOMSG | The queue does not contain a message of the desired type and (**msgtyp** & **IPC_NOWAIT**) is |
| EFAULT | *msgp* points to an illegal address. |
| EINTR | The call was interrupted by the delivery of a signal. |

SEE ALSO

**intro**(2), **msgctl**(2), **msgget**(2), **signal**(3)

## NAME

msync – synchronize memory with physical storage

## SYNOPSIS

**#include <sys/mman.h>**
**msync(addr, len, flags)**
**caddr_t addr; int len, flags;**

## DESCRIPTION

**msync**( ) writes all modified copies of pages over the range [*addr, addr + len*) to their permanent storage locations. **msync**( ) optionally invalidates any copies so that further references to the pages will be obtained by the system from their permanent storage locations.

Values for *flags* are defined in **<sys/mman.h>** as:

```
#define MS_ASYNC        0x1     /* Return immediately */
#define MS_INVALIDATE   0x2     /* Invalidate mappings */
```

and are used to control the behavior of **msync**. One or more flags may be specified in a single call.

MS_ASYNC returns **msync**( ) immediately once all I/O operations are scheduled; normally, **msync**( ) will not return until all I/O operations are complete. MS_INVALIDATE invalidates all cached copies of data from memory objects, requiring them to be re-obtained from the object's permanent storage location upon the next reference.

**msync**( ) should be used by programs which require a memory object to be in a known state, for example in building transaction facilities.

## RETURN VALUE

A 0 value is returned on success. A -1 value indicates an error.

## ERRORS

**msync**( ) fails if:

| | |
|---|---|
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ENOMEM | Addresses in the range [*addr, addr + len*) are outside the valid range for the address space of a process. |
| EINVAL | Either *addr* is not a multiple of the current page size, or *len* is negative. |
| EINVAL | One of the flags MS_ASYNC or MS_INVALID is invalid. |

## NAME

munmap – unmap pages of memory.

## SYNOPSIS

```
#include <sys/mman.h>
munmap(addr, len)
caddr_t addr; int len;
```

## DESCRIPTION

**munmap()** removes the mappings for pages in the range [*addr*, *addr* + *len*). Further references to these pages will result in the delivery of a SIGSEGV signal to the process, unless these pages are considered part of the "data" or "stack" segments.

**brk()** and **mmap()** often perform implicit **munmap**'s.

## RETURN VALUE

**munmap()** returns 0 on success, −1 on failure.

## ERRORS

**munmap()** will fail if:

| | |
|---|---|
| EINVAL | *addr* is not a multiple of the page size as returned by **getpagesize**(2). |
| EINVAL | Addresses in the range [*addr*, *addr* + *len*) are outside the valid range for the address space of a process. |

## SEE ALSO

**brk**(2), **getpagesize**(2), **mmap**(2)

**NAME**

nfssvc, async_daemon – NFS daemons

**SYNOPSIS**

**nfssvc (sock)**
**int sock;**

**async_daemon()**

**DESCRIPTION**

**nfssvc()** starts an NFS daemon listening on socket *sock*. The socket must be **AF_INET**, and **SOCK_DGRAM** (protocol **UDP/IP**). The system call will return only if the socket is invalid.

**async_daemon()** implements the NFS daemon that handles asynchronous I/O for an NFS client. This system call never returns.

Both system calls result in kernel-only processes with user memory discarded.

**SEE ALSO**

**mountd**(8C)

**BUGS**

There should be a way to dynamically create kernel-only processes instead of having to make system calls from userland to simulate this.

NAME

open – open or create a file for reading or writing

SYNOPSIS

**#include <fcntl.h>**

**int open(path, flags [ , mode ] )**
**char *path;**
**int flags, mode;**

DESCRIPTION

*path* points to the pathname of a file. **open()** opens the named file for reading and/or writing, as specified by the *flags* argument, and returns a descriptor for that file. The *flags* argument may indicate the file is to be created if it does not already exist (by specifying the O_CREAT flag), in which case the file is created with mode *mode* as described in **chmod**(2) and modified by the process' umask value (see **umask**(2)). If the path is a null string, the kernel maps this null pathname to '.', the current directory. *flags* values are constructed by ORing flags from the following list (only one of the first three flags below may be used):

**O_RDONLY** Open for reading only.

**O_WRONLY** Open for writing only.

**O_RDWR** Open for reading and writing.

**O_NDELAY** When opening a FIFO with **O_RDONLY** or **O_WRONLY** set:

If **O_NDELAY** is set:

> An **open()** for reading-only will return without delay. An **open()** for writing-only will return an error if no process currently has the file open for reading.

If **O_NDELAY** is clear:

> An **open()** for reading-only will block until a process opens the file for writing. An **open()** for writing-only will block until a process opens the file for reading.

When opening a file associated with a communication line:

If **O_NDELAY** is set:

> The open will return without waiting for carrier. The first time the process attempts to perform I/O on the open file it will block (not currently implemented).

If **O_NDELAY** is clear:

> The open will block until carrier is present.

**O_SYNC** When opening a regular file, this flag affects subsequent writes. If set, each **write**(2V) will wait for both the file data and file status to be physically updated.

**O_APPEND** If set, the file pointer will be set to the end of the file prior to each write.

**O_CREAT** If the file exists, this flag has no effect. Otherwise, the owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to either:

- the effective group ID of the process, if the filesystem was not mounted with the BSD file-creation semantics flag (see **mount**(2)) and the set-gid bit of the parent directory is clear, or
- the group ID of the directory in which the file is created.

The low-order 12 bits of the file mode are set to the value of *mode*, modified as follows (see **creat**(2)):

- All bits set in the file mode creation mask of the process are cleared. See **umask**(2).
- The "save text image after execution" bit of the mode is cleared. See **chmod**(2).
- The "set group ID on execution" bit of the mode is cleared if the effective user ID of the process is not super-user and the process is not a member of the group of the created file.

**O_TRUNC** If the file exists, its length is truncated to 0 and the mode and owner are unchanged.

**O_EXCL** If **O_EXCL** and **O_CREAT** are set, **open()** will fail if the file exists. This can be used to implement a simple exclusive access locking mechanism. If **O_EXCL** is set and the last component of the pathname is a symbolic link, the open will fail even if the symbolic link points to a non-existent name.

The file pointer used to mark the current position within the file is set to the beginning of the file.

The new descriptor is set to remain open across **execve**(2) system calls; see **close**(2) and **fcntl**(2V).

There is a system enforced limit on the number of open file descriptors per process, whose value is returned by the **getdtablesize**(2) call.

## SYSTEM V DESCRIPTION

If the **O_NDELAY** flag is set on an **open**, that flag is set for that file descriptor (see **fcntl**(2V) and may affect subsequent reads and writes. See **read**(2V) and **write**(2V).

## RETURN VALUE

The value −1 is returned if an error occurs, and external variable **errno** is set to indicate the cause of the error. Otherwise a non-negative numbered file descriptor for the new open file is returned.

## ERRORS

**open()** fails if:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | **O_CREAT** is not set and the named file does not exist. |
| ENOENT | A component of the path prefix of *path* does not exist. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| EACCES | The required permissions (for reading and/or writing) are denied for the file named by *path*. |
| EACCES | The file referred to by *path* does not exist, **O_CREAT** is specified, and the directory in which it is to be created does not permit writing. |
| EISDIR | The named file is a directory, and the arguments specify it is to be opened for writing. |
| ENXIO | **O_NDELAY** is set, the named file is a FIFO, **O_WRONLY** is set, and no process has the file open for reading. |
| EMFILE | The system limit for open file descriptors per process has already been reached. |
| ENFILE | The system file table is full. |

| | |
|---|---|
| ENOSPC | The file does not exist, **O_CREAT** is specified, and the directory in which the entry for the new file is being placed cannot be extended because there is no space left on the file system containing the directory. |
| ENOSPC | The file does not exist, **O_CREAT** is specified, and there are no free inodes on the file system on which the file is being created. |
| EDQUOT | The file does not exist, **O_CREAT** is specified, and the directory in which the entry for the new file is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted. |
| EDQUOT | The file does not exist, **O_CREAT** is specified, and the user's quota of inodes on the file system on which the file is being created has been exhausted. |
| EROFS | The named file does not exist, **O_CREAT** is specified, and the file system on which it is to be created is a read-only file system. |
| EROFS | The named file resides on a read-only file system, and the file is to be opened for writing. |
| ENXIO | The file is a character special or block special file, and the associated device does not exist. |
| EINTR | A signal was caught during the **open( )** system call. |
| EIO | A hangup or error occurred during a STREAMS **open**. |
| ENOSR | A *stream* could not be allocated. |
| ENXIO | A STREAMS module or driver open routine failed. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EFAULT | *path* points outside the process's allocated address space. |
| EEXIST | **O_EXCL** and **O_CREAT** were both specified and the file exists. |
| EOPNOTSUPP | An attempt was made to open a socket (not currently implemented). |

SEE ALSO

**chmod**(2), **close**(2), **creat**(2), **dup**(2), **fcntl**(2V), **getdtablesize**(2), **getmsg**(2), **lseek**(2), **mount**(2), **putmsg**(2), **read**(2V), **umask**(2), **write**(2V)

## NAME

pipe – create an interprocess communication channel

## SYNOPSIS

**pipe(fildes)**
**int fildes[2];**

## DESCRIPTION

The **pipe()** system call creates an I/O mechanism called a pipe and returns two file descriptors, *fildes*[0] and *fildes*[1]. *fildes*[0] is opened for reading and *fildes*[1] is opened for writing. When the pipe is written using the descriptor *fildes*[1] up to 4096 bytes of data are buffered before the writing process is blocked. A read only file descriptor *fildes*[0] accesses the data written to *fildes*[1] on a FIFO **(first-in-first-out)**basis.

It is assumed that after the pipe has been set up, two (or more) cooperating processes (created by subsequent **fork()** calls) will pass data through the pipe with **read()** and **write()** calls.

The shell has a syntax to set up a linear array of processes connected by pipes.

Read calls on an empty pipe (no buffered data) with only one end (all write file descriptors closed) returns an end-of-file.

Pipes are really a special case of the **socketpair**(2) call and, in fact, are implemented as such in the system.

A signal is generated if a write on a pipe with only one end is attempted.

## RETURN VALUE

The function value zero is returned if the pipe was created; –1 if an error occurred.

## ERRORS

The **pipe()** call will fail if:

| | |
|---|---|
| EMFILE | Too many descriptors are active. |
| ENFILE | The system file table is full. |
| EFAULT | The *fildes* file descriptor pair is in an invalid area of the process's address space. |

## SEE ALSO

**sh**(1), **fork**(2), **read**(2V), **socketpair**(2), **write**(2V)

## BUGS

Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.

NAME

poll – STREAMS I/O multiplexing

SYNOPSIS

```
#include <stropts.h>
#include <poll.h>

int poll(fds, nfds, timeout)
struct pollfd *fds;
unsigned long nfds;
int timeout;
```

DESCRIPTION

**poll**( ) provides users with a mechanism for multiplexing input/output over a set of file descriptors that reference open streams (see **intro**(2)). **poll**( ) identifies those streams on which a user can send or receive messages, or on which certain events have occurred. A user can receive messages using **read**(2V) or **getmsg**(2) and can send messages using **write**(2V) and **putmsg**(2). Certain **ioctl**(2) calls, such as **I_RECVFD** and **I_SENDFD** (see **streamio**(4)), can also be used to receive and send messages.

*fds* specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one element for each open file descriptor of interest. The array's elements are **pollfd** structures which contain the following members:

```
int fd;          /* file descriptor */
short events;    /* requested events */
short revents;   /* returned events */
```

where **fd** specifies an open file descriptor and **events** and **revents** are bitmasks constructed by ORing any combination of the following event flags:

**POLLIN** A non-priority or file descriptor passing message (see **I_RECVFD**) is present on the **stream head** read queue. This flag is set even if the message is of zero length. In **revents**, this flag is mutually exclusive with **POLLPRI**.

**POLLPRI** A priority message is present on the **stream head** read queue. This flag is set even if the message is of zero length. In **revents**, this flag is mutually exclusive with **POLLIN**.

**POLLOUT** The first downstream write queue in the *stream* is not full. Priority control messages can be sent (see **putmsg**(2)) at any time.

**POLLERR** An error message has arrived at the **stream head**. This flag is only valid in the **revents** bitmask; it is not used in the **events** field.

**POLLHUP** A hangup has occurred on the *stream*. This event and **POLLOUT** are mutually exclusive; a *stream* can never be writable if a hangup has occurred. However, this event and **POLLIN** or **POLLPRI** are not mutually exclusive. This flag is only valid in the **revents** bitmask; it is not used in the *events* field.

**POLLNVAL** The specified **fd** value does not belong to an open *stream*. This flag is only valid in the **revents** field; it is not used in the **events** field.

For each element of the array pointed to by *fds*, **poll**( ) examines the given file descriptor for the **event**(s) specified in **events**. The number of file descriptors to be examined is specified by *nfds*. If *nfds* exceeds the system limit of open files (see **getdtablesize**(2)), **poll**( ) will fail.

If the value **fd** is less than zero, **events** is ignored and **revents** is set to 0 in that entry on return from **poll**.

The results of the **poll**( ) query are stored in the **revents** field in the **pollfd** structure. Bits are set in the **revents** bitmask to indicate which of the requested events are true. If none are true, none of the specified bits is set in **revents** when the **poll**( ) call returns. The event flags **POLLHUP**, **POLLERR**, and **POLLNVAL** are always set in **revents** if the conditions they indicate are true; this occurs even though these flags were not present in **events**.

If none of the defined events have occurred on any selected file descriptor, **poll()** waits at least *timeout* milliseconds for an event to occur on any of the selected file descriptors. On a computer where millisecond timing accuracy is not available, *timeout* is rounded up to the nearest legal value available on that system. If the value *timeout* is 0, **poll()** returns immediately. If the value of *timeout* is −1, **poll()** blocks until a requested event occurs or until the call is interrupted. **poll()** is not affected by the **O_NDELAY** flag.

## RETURN VALUE

Upon successful completion, a non-negative value is returned. A positive value indicates the total number of file descriptors that has been selected (for instance, file descriptors for which the **revents** field is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**poll()** fails if one or more of the following are true:

| | |
|---|---|
| EAGAIN | Allocation of internal data structures failed, but the request should be attempted again. |
| EFAULT | Some argument points outside the allocated address space. |
| EINTR | A signal was caught during the **poll()** system call. |
| EINVAL | The argument *nfds* is less than zero, or *nfds* is greater than the system limit of open files. |

## SEE ALSO

**getdtablesize**(2), **getmsg**(2), **intro**(2), **ioctl**(2), **putmsg**(2), **read**(2V), **write**(2V), **streamio**(4)

NAME

profil – execution time profile

SYNOPSIS

**profil(buff, bufsiz, offset, scale)**
**char *buff;**
**int bufsiz, offset, scale;**

DESCRIPTION

**profil( )** enables run-time execution profiling, and reserves a buffer for maintaining raw profiling statistics. *buff* points to an area of core of length *bufsiz* (in bytes). After the call to **profil**, the user's program counter (pc) is examined at each clock tick (10 milliseconds on Sun-4 systems, 20 milliseconds on Sun-2 and Sun-3 systems); *offset* is subtracted from its value, and the result multiplied by *scale*. If the resulting number corresponds to a word within the buffer, that word is incremented.

*scale* is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0x10000 gives a 1-to-1 mapping of pc values to words in *buff*; 0x8000 maps each pair of instruction words together. 0x2 maps all instructions onto the beginning of *buff* (producing a non-interrupting core clock).

Profiling is turned off by giving a *scale* of 0 or 1. It is rendered ineffective by giving a *bufsiz* of 0. Profiling is turned off when an **execve( )** is executed, but remains on in child and parent both after a **fork( )**. Profiling is turned off if an update in *buff* would cause a memory fault.

RETURN VALUE

A 0, indicating success, is always returned.

SEE ALSO

**gprof**(1), **getitimer**(2), **monitor**(3)

NAME

ptrace – process trace

SYNOPSIS

```
#include <signal.h>
#include <sys/ptrace.h>
#include <sys/wait.h>

ptrace(request, pid, addr, data [ , addr2 ] )
enum ptracereq request;
int pid;
char *addr;
int data;
char *addr2;
```

DESCRIPTION

**ptrace()** provides a means by which a process may control the execution of another process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging. There are five arguments whose interpretation depends on the *request* argument. Generally, *pid* is the process ID of the traced process. A process being traced behaves normally until it encounters some signal whether internally generated like "illegal instruction" or externally generated like "interrupt". See **sigvec**(2) for the list. Then the traced process enters a stopped state and the tracing process is notified using **wait**(2). When the traced process is in the stopped state, its core image can be examined and modified using **ptrace**. If desired, another **ptrace()** request can then cause the traced process either to terminate or to continue, possibly ignoring the signal.

Note: several different values of the *request* argument can make **ptrace()** return data values — since −1 is a possibly legitimate value, to differentiate between −1 as a legitimate value and −1 as an error code, you should clear the **errno** global error code before doing a **ptrace()** call, and then check the value of **errno** afterwards.

The value of the *request* argument determines the precise action of the call:

**PTRACE_TRACEME**

This request is the only one used by the traced process; it declares that the process is to be traced by its parent. All the other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

**PTRACE_PEEKTEXT**
**PTRACE_PEEKDATA**

The word in the traced process's address space at *addr* is returned. If the instruction and data spaces are separate (for example, historically on a PDP-11), request **PTRACE_PEEKTEXT** indicates instruction space while **PTRACE_PEEKDATA** indicates data space. Otherwise, either request may be used, with equal results; *addr* must be even on a Sun-2 system or a multiple of 4 on a Sun-4 system. The child must be stopped. The input *data* and *addr2* are ignored.

**PTRACE_PEEKUSER**

The word of the system's per-process data area corresponding to *addr* is returned. *addr* must be a valid offset within the kernel's per-process data pages. This space contains the registers and other information about the process; its layout corresponds to the *user* structure in the system (see **<sys/user.h>**).

**PTRACE_POKETEXT**
**PTRACE_POKEDATA**

The given *data* is written at the word in the process's address space corresponding to *addr*. *addr* must be even on a Sun-2 system or a multiple of 4 on a Sun-4 system. No useful value is returned. If the instruction and data spaces are separate, request **PTRACE_PEEKTEXT** indicates instruction space while **PTRACE_PEEKDATA** indicates data space. The **PTRACE_POKETEXT** request must be used to write into a process's text space even if the instruction and data spaces are not separate.

**PTRACE_POKEUSER**
The process's system data is written, as it is read with request **PTRACE_PEEKUSER**. Only a few locations can be written in this way: the general registers, the floating point status and registers, and certain bits of the processor status word.

**PTRACE_CONT**
The *data* argument is taken as a signal number and the child's execution continues at location *addr* as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal that caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop. If *addr* is (int *)1 then execution continues from where it stopped. *addr* must be a multiple of 4 on a Sun-4 system.

**PTRACE_KILL**
The traced process terminates, with the same consequences as **exit**(2).

**PTRACE_SINGLESTEP**
Execution continues as in request **PTRACE_CONT**; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is **SIGTRAP**. On Sun-2, Sun-3, and Sun386i systems, the status register T-bit is used and just one instruction is executed. This is part of the mechanism for implementing breakpoints. On a Sun-4 system this will return an error since there is no hardware assist for this feature. Instead, the user should insert breakpoint traps in the debugged program with **PTRACE_POKETEXT**.

**PTRACE_ATTACH**
Attach to the process identified by the *pid* argument and begin tracing it. Process *pid* does not have to be a child of the requestor, but the requestor must have permission to send process *pid* a signal and the effective user IDs of the requesting process and process *pid* must match.

**PTRACE_DETACH**
Detach the process being traced. Process *pid* is no longer being traced and continues its execution. The *data* argument is taken as a signal number and the process continues at location *addr* as if it had incurred that signal.

**PTRACE_GETREGS**
The traced process's registers are returned in a structure pointed to by the *addr* argument. The registers include the general purpose registers, the program counter and the program status word. The "regs" structure defined in **<machine/reg.h>** describes the data that is returned.

**PTRACE_SETREGS**
The traced process's registers are written from a structure pointed to by the *addr* argument. The registers include the general purpose registers, the program counter and the program status word. The "regs" structure defined in **<machine/reg.h>** describes the data that is set.

**PTRACE_GETFPREGS**
(Sun-3, Sun-4 and Sun386i systems only) The traced process's FPP status is returned in a structure pointed to by the *addr* argument. The status includes the 68881 (80387 on Sun386i systems) floating point registers and the control, status, and instruction address registers. The "fp_status" structure defined in **<machine/reg.h>** describes the data that is returned. On Sun-2 systems this will return an error since there is no user visible floating point state. The **fp_state** structure defined in **<machine/fp.h>** describes the data that is returned on a Sun386i system.

**PTRACE_SETFPREGS**
(Sun-3, Sun-4 and Sun386i systems only) The traced process's FPP status is written from a structure pointed to by the *addr* argument. The status includes the FPP floating point registers and the control, status, and instruction address registers. The "fp_status" structure defined in **<machine/reg.h>** describes the data that is set. On Sun-2 systems this will return an error since there is no user visible floating point state. The "fp_state" structure defined in <machine/fp.h> describes the data that is returned on a Sun386i system.

**PTRACE_GETFPAREGS**
(a Sun-3 system with FPA only) The traced process's FPA registers are returned in a structure pointed to by the *addr* argument. The "fpa_regs" structure defined in **<machine/reg.h>** describes the data that is returned.

**PTRACE_SETFPAREGS**
(a Sun-3 system with FPA only) The traced process's FPA registers are written from a structure pointed to by the *addr* argument. The "fpa_regs" structure defined in **<machine/reg.h>** describes the data that is set.

**PTRACE_READTEXT**
**PTRACE_READDATA**
Read data from the address space of the traced process. If the instruction and data spaces are separate, request **PTRACE_READTEXT** indicates instruction space while **PTRACE_READDATA** indicates data space. The *addr* argument is the address within the traced process from where the data is read, the *data* argument is the number of bytes to read, and the *addr2* argument is the address within the requesting process where the data is written.

**PTRACE_WRITETEXT**
**PTRACE_WRITEDATA**
Write data into the address space of the traced process. If the instruction and data spaces are separate, request **PTRACE_READTEXT** indicates instruction space while **PTRACE_READDATA** indicates data space. The *addr* argument is the address within the traced process where the data is written, the *data* argument is the number of bytes to write, and the *addr2* argument is the address within the requesting process from where the data is read.

**PTRACE_SETWRBKPT**
(Sun386i systems only) Set a write breakpoint at location *addr* in the process being traced. Whenever a write is directed to this location a breakpoint will occur and a SIGTRAP signal will be sent to the process. The *data* argument specifies which debug register should be used for the address of the breakpoint and must be in the range 0 through 3, inclusive. The *addr2* argument specifies the length of the operand in bytes, and must be one of 1, 2, or 4.

**PTRACE_SETACBKPT**
(Sun386i systems only) Set an access breakpoint at location *addr* in the process being traced. When location *addr* is read or written a breakpoint will occur and the process will be sent a SIGTRAP signal. The *data* argument specifies which debug register should be used for the address of the breakpoint and must be in the range 0 through 3, inclusive. The *addr2* argument specifies the length of the operand in bytes, and must be one of 1, 2, or 4.

**PTRACE_CLRBKPT**
(Sun386i systems only) Clears all break points set with **PTRACE_SETACBKPT** or **PTRACE_SETWRBKPT**.

**PTRACE_SYSCALL**
Execution continues as in request **PTRACE_CONT**; until the process makes a system call. The process receives a **SIGTRAP** signal and stops. At this point the arguments to the system call may be inspected in the process *user* structure using the **PTRACE_PEEKUSER** request. The system call number is available in place of the 8th argument. Continuing with another **PTRACE_SYSCALL** will stop the process again at the completion of the system call. At this point the result of the system call and error value may be inspected in the process *user* structure.

**PTRACE_DUMPCORE**
Dumps a core image of the traced process to a file. The name of the file is obtained from the *addr* argument.

As indicated, these calls (except for requests **PTRACE_TRACEME**, **PTRACE_ATTACH** and **PTRACE_DETACH**) can be used only when the subject process has stopped. The **wait()** call is used to determine when a process stops; in such a case the "termination" status returned by **wait()** has the value

**WSTOPPED** to indicate a stop rather than genuine termination.

To forestall possible fraud, **ptrace()** inhibits the setUID and setGID facilities on subsequent **execve**(2) calls. If a traced process calls **execve**, it will stop before executing the first instruction of the new image, showing signal **SIGTRAP**.

On the Sun, "word" also means a 32-bit integer.

## RETURN VALUE

In general, a 0 value is returned if the call succeeds. Note: this is not always true because requests such as **PTRACE_PEEKTEXT** and **PTRACE_PEEKDATA** return legitimate values. If the call fails then a −1 is returned and the global variable **errno** is set to indicate the error.

## ERRORS

| | |
|---|---|
| EIO | The request code is invalid. |
| ESRCH | The specified process does not exist. |
| ESRCH | The request requires the process to be one which is traced by the current process and stopped, but it is not stopped or it is not being traced by the current process. |
| EIO | The given signal number is invalid. |
| EIO | The specified address is out of bounds. |
| EPERM | The specified process cannot be traced. |

## SEE ALSO

**adb**(1), **intro**(2), **ioctl**(2), **sigvec**(2), **wait**(2)

## BUGS

**ptrace()** is unique and arcane; it should be replaced with a special file which can be opened and read and written. The control functions could then be implemented with **ioctl**(2) calls on this file. This would be simpler to understand and have much higher performance.

The requests **PTRACE_TRACEME** through **PTRACE_SINGLESTEP** are standard UNIX system **ptrace()** requests. The requests **PTRACE_ATTACH** through **PTRACE_DUMPCORE** and the fifth argument, *addr2*, are unique to SunOS.

The request **PTRACE_TRACEME** should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use "illegal instruction" signals at a very high rate) could be efficiently debugged.

The error indication, −1, is a legitimate function value; **errno**, (see **intro**(2)), can be used to clarify what it means.

## NAME

putmsg – send a message on a stream

## SYNOPSIS

**#include <stropts.h>**

**int putmsg (fd, ctlptr, dataptr, flags)**

**int fd;**

**struct strbuf *ctlptr;**

**struct strbuf *dataptr;**

**int flags;**

## DESCRIPTION

**putmsg()** creates a message (see **intro**(2)) from user specified buffer(s) and sends the message to a STREAMS file. The message may contain either a data part, a control part or both. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The semantics of each part is defined by the STREAMS module that receives the message.

*fd* specifies a file descriptor referencing an open *stream*. *ctlptr* and *dataptr* each point to a **strbuf** structure that contains the following members:

```
int maxlen; /* not used */
int len;    /* length of data */
char *buf;  /* ptr to buffer */
```

*ctlptr* points to the structure describing the control part, if any, to be included in the message. The **buf** field in the **strbuf** structure points to the buffer where the control information resides, and the **len** field indicates the number of bytes to be sent. The **maxlen** field is not used in **putmsg()** (see **getmsg**(2)). In a similar manner, *dataptr* specifies the data, if any, to be included in the message. *flags* may be set to the values 0 or **RS_HIPRI** and is used as described below.

To send the data part of a message, *dataptr* must not be a NULL pointer and the **len** field of *dataptr* must have a value of 0 or greater. To send the control part of a message, the corresponding values must be set for *ctlptr*. No data (control) part will be sent if either *dataptr* (*ctlptr*) is a NULL pointer or the **len** field of *dataptr* (*ctlptr*) is set to –1.

If a control part is specified, and *flags* is set to **RS_HIPRI**, a *priority* message is sent. If *flags* is set to 0, a non-priority message is sent. If no control part is specified, and *flags* is set to **RS_HIPRI**, **putmsg()** fails and sets **errno** to EINVAL. If no control part and no data part are specified, and *flags* is set to 0, no message is sent, and 0 is returned.

For non-priority messages, **putmsg()** will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, **putmsg()** does not block on this condition. For non-priority messages, **putmsg()** does not block when the write queue is full and **O_NDELAY** is set. Instead, it fails and sets **errno** to EAGAIN.

**putmsg()** also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether **O_NDELAY** has been specified. No partial message is sent.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and **errno** is set to indicate the error.

## ERRORS

**putmsg()** fails if one or more of the following are true:

EAGAIN — A non-priority message was specified, the **O_NDELAY** flag is set and the *stream* write queue is full due to internal flow control conditions.

| | |
|---|---|
| EAGAIN | Buffers could not be allocated for the message that was to be created. |
| EBADF | *fd* is not a valid file descriptor open for writing. |
| EFAULT | *ctlptr* or *dataptr* points outside the allocated address space. |
| EINTR | A signal was caught during the **putmsg( )** system call. |
| EINVAL | An undefined value was specified in *flags*, or *flags* is set to **RS_HIPRI** and no control part was supplied. |
| EINVAL | The *stream* referenced by *fd* is linked below a multiplexor. |
| ENOSTR | A *stream* is not associated with *fd*. |
| ENXIO | A hangup condition was generated downstream for the specified *stream*. |
| ERANGE | The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message. |

A **putmsg( )** also fails if a STREAMS error message had been processed by the *stream* head before the call to **putmsg**. The error returned is the value contained in the STREAMS error message.

SEE ALSO

**intro**(2), **getmsg**(2), **poll**(2), **read**(2V), **write**(2V)

## NAME

quotactl – manipulate disk quotas

## SYNOPSIS

**#include <ufs/quota.h>**

**int quotactl(cmd, special, uid, addr)**
**int cmd;**
**char *special;**
**int uid;**
**caddr_t addr;**

## DESCRIPTION

The **quotactl()** call manipulates disk quotas. *cmd* indicates a command to be applied to the user ID *uid*. *special* is a pointer to a null-terminated string containing the path name of the block special device for the file system being manipulated. The block special device must be mounted as a UFS file system (see **mount**(2)). *addr* is the address of an optional, command specific, data structure which is copied in or out of the system. The interpretation of *addr* is given with each command below.

Q_QUOTAON
: Turn on quotas for a file system. *addr* points to the path name of file containing the quotas for the file system. The quota file must exist; it is normally created with the **quotacheck**(8) program. This call is restricted to the super-user.

Q_QUOTAOFF
: Turn off quotas for a file system. *addr* and *uid* are ignored. This call is restricted to the super-user.

Q_GETQUOTA
: Get disk quota limits and current usage for user *uid*. *addr* is a pointer to a **dqblk** structure (defined in **<ufs/quota.h>**). Only the super-user may get the quotas of a user other than himself.

Q_SETQUOTA
: Set disk quota limits and current usage for user *uid*. *addr* is a pointer to a **dqblk** structure (defined in **<ufs/quota.h>**). This call is restricted to the super-user.

Q_SETQLIM
: Set disk quota limits for user *uid*. *addr* is a pointer to a **dqblk** structure (defined in **<ufs/quota.h>**). This call is restricted to the super-user.

Q_SYNC
: Update the on-disk copy of quota usages for a file system. If *special* is null then all file systems with active quotas are sync'ed. *addr* and *uid* are ignored.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

A **quotactl()** call will fail when one of the following occurs:

EINVAL
: The kernel has not been compiled with the QUOTA option.

EINVAL
: *cmd* is invalid.

ESRCH No disc quota is found for the indicated user or quotas have not been turned on for this file system.

EPERM The call is privileged and the caller was not the super-user.

ENODEV
: *special* is not a mounted UFS file system.

ENOTBLK
*special* is not a block device.

EACCES
(**Q_QUOTAON**) The quota file pointed to by *addr* exists but is either not a regular file or is not on the file system pointed to by *special*.

EBUSY **Q_QUOTAON** attempted while another **Q_QUOTAON** or **Q_QUOTAOFF** is in progress.

EUSERS
The quota table is full.

ENOENT
The file specified by *special* or *addr* does not exist.

EFAULT
*addr* or *special* are invalid.

**SEE ALSO**

**quota**(1), **getrlimit**(2), **mount**(2), **quotacheck**(8), **quotaon**(8)

**BUGS**

There should be some way to integrate this call with the resource limit interface provided by **setrlimit** and **getrlimit**(2).

Incompatible with Melbourne quotas.

NAME

read, readv – read input

SYNOPSIS

```
int read(d, buf, nbytes)
int d;
char *buf;
int nbytes;

#include <sys/types.h>
#include <sys/uio.h>

int readv(d, iov, iovcnt)
int d;
struct iovec *iov;
int iovcnt;
```

DESCRIPTION

**read()** attempts to read *nbytes* of data from the object referenced by the descriptor *d* into the buffer pointed to by *buf*. **readv()** performs the same action, but scatters the input data into the *iovcnt* buffers specified by the members of the *iov* array: iov[0], iov[1], ..., iov[iovcnt–1].

For **readv**, the *iovec* structure is defined as

```
struct iovec {
        caddr_t iov_base;
        int     iov_len;
};
```

Each *iovec* entry specifies the base address and length of an area in memory where data should be placed. **readv()** will always fill an area completely before proceeding to the next.

On objects capable of seeking, the **read()** starts at a position given by the pointer associated with *d* (see **lseek**(2)). Upon return from **read**, the pointer is incremented by the number of bytes actually read.

Objects that are not capable of seeking always read from the current position. The value of the pointer associated with such an object is undefined.

Upon successful completion, **read()** and **readv()** return the number of bytes actually read and placed in the buffer. The system guarantees to read the number of bytes requested if the descriptor references a normal file which has that many bytes left before the end-of-file, but in no other case.

If the returned value is 0, then end-of-file has been reached.

A **read()** or **readv()** from a STREAMS (see **Intro**(2)) file can operate in three different modes: "byte-stream" mode, "message-nondiscard" mode, and "message-discard" mode. The default is byte-stream mode. This can be changed using the **I_SRDOPT ioctl()** request (see **streamio**(4)), and can be tested with the **I_GRDOPT ioctl**. In byte-stream mode, **read()** and **readv()** will retrieve data from the *stream* until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, **read()** and **readv()** will retrieve data until as many bytes as were requested are transferred, or until a message boundary is reached. If the **read()** or **readv()** does not retrieve all the data in a message, the remaining data are left on the *stream*, and can be retrieved by the next **read**, **readv**, or **getmsg**(2) call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the **read()** or **readv()** returns are discarded, and are not available for a subsequent **read**, **readv**, or *getmsg*.

When attempting to read from a descriptor associated with an empty pipe, socket, FIFO, or *stream*:

If the object the descriptor is associated with is marked for 4.2BSD-style non-blocking I/O (with the FIONBIO **ioctl** (2), or an **fcntl()** using the FNDELAY flag from **<sys/file.h>** or the **O_NDELAY**

flag from **<sys/fcntl.h>** in the 4.2BSD environment), the read will return −1 and **errno** will be set to EWOULDBLOCK.

If the descriptor is marked for System V-style non-blocking I/O (with an **fcntl( )** using the FNBIO flag from **<sys/file.h>** or the **O_NDELAY** flag from **<sys/fcntl.h>** in the System V environment), and does not refer to a *stream*, the read will return 0. Note: this is indistinguishable from end-of-file.

If the descriptor is marked for System V-style non-blocking I/O, and refers to a *stream*, the read will return −1 and **errno** will be set to EAGAIN.

If neither the descriptor nor the object it refers to are marked for non-blocking I/O, the read will block until data is available to be read or the object is has been "disconnected". A pipe or FIFO is "disconnected" when no process has the object open for writing; a socket that was connected is "disconnected" when the connection is broken; a stream is "disconnected" when a hangup condition occurs (for instance, when carrier drops on a terminal).

If the descriptor or the object is marked for non-blocking I/O, and less data are available than are requested by the **read( )** or **readv**, only the data that are available are returned, and the count indicates how many bytes of data were actually read.

When reading from a STREAMS file, handling of zero-byte messages is determined by the current read mode setting. In byte-stream mode, **read( )** and **readv( )** accept data until as many bytes as were requested are transferred, or until there is no more data to read, or until a zero-byte message block is encountered. **read( )** and **readv( )** then return the number of bytes read, and places the zero-byte message back on the *stream* to be retrieved by the next **read**, **readv**, or **getmsg**. In the two other modes, a zero-byte message returns a value of 0 and the message is removed from the *stream*. When a zero-byte message is read as the first message on a *stream*, a value of 0 is returned regardless of the read mode.

A **read( )** or **readv( )** from a STREAMS file can only process data messages. It cannot process any type of protocol message and will fail if a protocol message is encountered at the **streamhead.**

## RETURN VALUE

If successful, the number of bytes actually read is returned. Otherwise, a −1 is returned and the global variable **errno** is set to indicate the error.

## ERRORS

**read( )** and **readv( )** will fail if one or more of the following are true:

| | |
|---|---|
| EBADF | *d* is not a valid file descriptor open for reading. |
| EISDIR | *d* refers to a directory which is on a file system mounted using the NFS. |
| EBADMSG | The message waiting to be read on a *stream* is not a data message. |
| EFAULT | *buf* points outside the allocated address space. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EINTR | A read from a slow device was interrupted before any data arrived by the delivery of a signal. |
| EINVAL | The *stream* is linked below a multiplexor. |
| EINVAL | The pointer associated with *d* was negative. |
| EWOULDBLOCK | The file was marked for 4.2BSD-style non-blocking I/O, and no data were ready to be read. |
| EAGAIN | The descriptor referred to a *stream*, was marked for System V-style non-blocking I/O, and no data were ready to be read. |
| | In addition, **readv( )** may return one of the following errors: |
| EINVAL | *iovcnt* was less than or equal to 0, or greater than 16. |

| | |
|---|---|
| EINVAL | One of the **iov_len** values in the *iov* array was negative. |
| EINVAL | The sum of the **iov_len** values in the *iov* array overflowed a 32-bit integer. |
| EFAULT | Part of *iov* points outside the process's allocated address space. |

A **read()** or **readv()** from a STREAMS file will also fail if an error message is received at the **stream**head. In this case, **errno** is set to the value returned in the error message. If a hangup occurs on the *stream* being read, **read()** will continue to operate normally until the **stream head** read queue is empty. Thereafter, it will return 0.

**SEE ALSO**

**dup**(2), **fcntl**(2V), **getmsg**(2), **intro**(2), **ioctl**(2), **lseek**(2), **open**(2V), **pipe**(2), **select**(2), **socket**(2), **socketpair**(2), **streamio**(4)

## NAME

readlink – read value of a symbolic link

## SYNOPSIS

**int readlink(path, buf, bufsiz)**
**char *path, *buf;**
**int bufsiz;**

## DESCRIPTION

**readlink()** places the contents of the symbolic link referred to by *path* in the buffer *buf* which has size *bufsiz*. The contents of the link are not null terminated when returned.

## RETURN VALUE

The call returns the count of characters placed in the buffer if it succeeds, or a −1 if an error occurs, placing the error code in the global variable **errno**.

## ERRORS

**readlink()** will fail and the buffer will be unchanged if:

| | |
|---|---|
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The named file does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EINVAL | The named file is not a symbolic link. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EFAULT | *path* or *buf* extends outside the process's allocated address space. |

## SEE ALSO

**stat**(2), **symlink**(2)

## NAME

reboot – reboot system or halt processor

## SYNOPSIS

**#include <sys/reboot.h>**

**int reboot(howto [ , bootargs ] )**
**int howto;**
**char *bootargs;**

## DESCRIPTION

**reboot()** reboots the system, and is invoked automatically in the event of unrecoverable system failures. *howto* is a mask of options passed to the bootstrap program. The system call interface permits only **RB_HALT** or **RB_AUTOBOOT** to be passed to the reboot program; the other flags are used in scripts stored on the console storage media, or used in manual bootstrap procedures. When none of these options (for instance **RB_AUTOBOOT**) is given, the system is rebooted from file **/vmunix** in the root file system of unit 0 of a disk chosen in a processor specific way. An automatic consistency check of the disks is then normally performed.

The bits of *howto* are:

**RB_HALT**
the processor is simply halted; no reboot takes place. **RB_HALT** should be used with caution.

**RB_ASKNAME**
Interpreted by the bootstrap program itself, causing it to inquire as to what file should be booted. Normally, the system is booted from the file "vmunix" without asking.

**RB_SINGLE**
Normally, the reboot procedure involves an automatic disk consistency check and then multi-user operations. **RB_SINGLE** prevents the consistency check, rather simply booting the system with a single-user shell on the console. **RB_SINGLE** is interpreted by the **init**(8) program in the newly booted system.

**RB_DUMP**
A system core dump is performed before rebooting.

**RB_STRING**
The optional argument *bootargs* is passed to the bootstrap program. See **boot**(8S) for details. This option overrides **RB_SINGLE** but the same effect can be achieved by including **–s** as an option in *bootargs*.

Only the super-user may **reboot()** a machine.

## RETURN VALUES

If successful, this call never returns. Otherwise, a –1 is returned and an error is returned in the global variable **errno**.

## ERRORS

EPERM The caller is not the super-user.

## FILES

**/vmunix**

## SEE ALSO

**crash**(8S), **halt**(8), **init**(8), **intro**(8), **reboot**(8)

NAME

recv, recvfrom, recvmsg – receive a message from a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(s, buf, len, flags)
int s;
char *buf;
int len, flags;

int recvfrom(s, buf, len, flags, from, fromlen)
int s;
char *buf;
int len, flags;
struct sockaddr *from;
int *fromlen;

int recvmsg(s, msg, flags)
int s;
struct msghdr *msg;
int flags;
```

DESCRIPTION

*s* is a socket created with **socket**(2). **recv**, **recvfrom**, and **recvmsg** are used to receive messages from another socket. **recv**( ) may be used only on a *connected* socket (see **connect**(2)), while **recvfrom**( ) and **recvmsg**( ) may be used to receive data on a socket whether it is in a connected state or not.

If *from* is not a NULL pointer, the source address of the message is filled in. *fromlen* is a value-result parameter, initialized to the size of the buffer associated with *from*, and modified on return to indicate the actual size of the address stored there. The length of the message is returned. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from (see **socket**(2)).

If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is nonblocking (see **ioctl**(2)) in which case −1 is returned with the external variable **errno** set to EWOULDBLOCK.

The **select**(2) call may be used to determine when more data arrives.

The *flags* parameter is formed by ORing one or more of the following:

**MSG_OOB**
: Read any "out-of-band" data present on the socket, rather than the regular "in-band" data.

**MSG_PEEK**
: "Peek" at the data present on the socket; the data is returned, but not consumed, so that a subsequent receive operation will see the same data.

The **recvmsg**( ) call uses a **msghdr** structure to minimize the number of directly supplied parameters. This structure is defined in **<sys/socket.h>**, and includes the following members:

```
caddr_t     msg_name;       /* optional address */
int         msg_namelen;    /* size of address */
struct iovec *msg_iov;      /* scatter/gather array */
int         msg_iovlen;     /* # elements in msg_iov */
caddr_t     msg_accrights;  /* access rights sent/received */
int         msg_accrightslen;
```

Here **msg_name** and **msg_namelen** specify the destination address if the socket is unconnected; **msg_name** may be given as a NULL pointer if no names are desired or required. The **msg_iov** and **msg_iovlen** describe the scatter-gather locations, as described in **read**(2V). A buffer to receive any access rights sent along with the message is specified in **msg_accrights**, which has length **msg_accrightslen**.

**RETURN VALUE**

These calls return the number of bytes received, or −1 if an error occurred.

**ERRORS**

The calls fail if:

| | |
|---|---|
| EBADF | *s* is an invalid descriptor. |
| ENOTSOCK | *s* is a descriptor for a file, not a socket. |
| EINTR | The operation was interrupted by delivery of a signal before any data was available to be received. |
| EFAULT | The data was specified to be received into a non-existent or protected part of the process address space. |
| EWOULDBLOCK | The socket is marked non-blocking and the requested operation would block. |

**SEE ALSO**

**connect**(2), **fcntl**(2V), **getsockopt**(2), **ioctl**(2), **read**(2V), **select**(2), **send**(2), **socket**(2)

## NAME

rename – change the name of a file

## SYNOPSIS

**int rename(from, to)**
**char *from, *to;**

## DESCRIPTION

**rename( )** renames the link named *from* as *to*. If *to* exists, then it is first removed. Both *from* and *to* must be of the same type (that is, both directories or both non-directories), and must reside on the same file system.

**rename( )** guarantees that an instance of *to* will always exist, even if the system should crash in the middle of the operation.

If the final component of *from* is a symbolic link, the symbolic link is renamed, not the file or directory to which it points.

### CAVEAT

The system can deadlock if a loop in the file system graph is present. This loop takes the form of an entry in directory **a**, say **a/file1**, being a hard link to directory **b**, and an entry in directory **b**, say **b/file2**, being a hard link to directory **a**. When such a loop exists and two separate processes attempt to perform '**rename a/file1 b/file2**' and '**rename b/file2 a/file1**', respectively, the system may deadlock attempting to lock both directories for modification. Hard links to directories should be replaced by symbolic links by the system administrator.

## RETURN VALUE

A 0 value is returned if the operation succeeds, otherwise **rename( )** returns −1 and the global variable **errno** indicates the reason for the failure.

## ERRORS

**rename( )** will fail and neither of the argument files will be affected if any of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of either *from* or *to* is not a directory. |
| ENAMETOOLONG | The length of a component of either *from* or *to* exceeds 255 characters, or the length of either *from* or *to* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of either *from* or *to* does not exist. |
| ENOENT | The file named by *from* does not exist. |
| EACCES | A component of the path prefix of either *from* or *to* denies search permission. |
| EACCES | The requested rename requires writing in a directory with a mode that denies write permission. |
| ELOOP | Too many symbolic links were encountered while translating either *from* or *to*. |
| EXDEV | The link named by *to* and the file named by *from* are on different logical devices (file systems). |
| ENOSPC | The directory in which the entry for the new name is being placed cannot be extended because there is no space left on the file system containing the directory. |
| EDQUOT | The directory in which the entry for the new name is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EROFS | The requested rename requires writing in a directory on a read-only file system. |
| EFAULT | Either or both of *from* or *to* point outside the process's allocated address space. |
| EINVAL | *from* is a parent directory of *to*, or an attempt is made to rename '.' or '..'. |

| | |
|---|---|
| ENOTEMPTY | *to* is a directory and is not empty. |
| EBUSY | *to* is a directory and is the mount point for a mounted file system. |

**SEE ALSO**

**open**(2V)

**NAME**

rmdir – remove a directory file

**SYNOPSIS**

**int rmdir(path)**
**char *path;**

**DESCRIPTION**

**rmdir()** removes a directory file whose name is given by *path*. The directory must not have any entries other than '.' and '..'.

**RETURN VALUE**

A 0 is returned if the remove succeeds; otherwise a −1 is returned and an error code is stored in the global location **errno**.

**ERRORS**

The named file is removed unless one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENOTDIR | The file referred to by *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The directory referred to by *path* does not exist. |
| EINVAL | The directory referred to by *path* is the current directory, '.'. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| ENOTEMPTY | The directory referred to by *path* contains files other than '.' and '..' in it. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| EACCES | Write permission is denied for the directory containing the link to be removed. |
| EBUSY | The directory to be removed is the mount point for a mounted file system. EIO An I/O error occurred while reading from or writing to the file system. |
| EROFS | The directory to be removed resides on a read-only file system. |
| EFAULT | *path* points outside the process's allocated address space. |

**SEE ALSO**

**mkdir**(2), **unlink**(2)

## NAME

select – synchronous I/O multiplexing

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/time.h>

int select (width, readfds, writefds, exceptfds, timeout)
int width;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;

FD_SET (fd, &fdset)
FD_CLR (fd, &fdset)
FD_ISSET (fd, &fdset)
FD_ZERO (&fdset)
int fd;
fd_set fdset;
```

## DESCRIPTION

**select()** examines the I/O descriptor sets whose addresses are passed in *readfds*, *writefds*, and *exceptfds* to see if some of their descriptors are ready for reading, ready for writing, or have an exceptional condition pending. *width* is the number of bits to be checked in each bit mask that represent a file descriptor; the descriptors from 0 through *width*−1 in the descriptor sets are examined. Typically *width* has the value returned by **getdtablesize**(2) for the maximum number of file descriptors. On return, **select()** replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned.

The descriptor sets are stored as bit fields in arrays of integers. The following macros are provided for manipulating such descriptor sets: **FD_ZERO** (*&fdset*) initializes a descriptor set *fdset* to the null set. **FD_SET**(*fd, &fdset* ) includes a particular descriptor *fd* in *fdset*. **FD_CLR**(*fd, &fdset*) removes *fd* from *fdset*. **FD_ISSET**(*fd, &fdset*) is nonzero if *fd* is a member of *fdset*, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to **FD_SETSIZE**, which is normally at least equal to the maximum number of descriptors supported by the system.

If *timeout* is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If *timeout* is a NULL pointer, the select blocks indefinitely. To effect a poll, the *timeout* argument should be a non-NULL pointer, pointing to a zero-valued **timeval** structure.

Any of *readfds*, *writefds*, and *exceptfds* may be given as NULL pointers if no descriptors are of interest.

Using **select()** to open a socket for reading is analogous to performing an **accept**(2) call.

## RETURN VALUE

**select()** returns the number of ready descriptors that are contained in the descriptor sets, or −1 if an error occurred. If the time limit expires then **select()** returns 0. If **select()** returns with an error, including one due to an interrupted call, the descriptor sets will be unmodified.

## ERRORS

An error return from **select()** indicates:

| | |
|---|---|
| EBADF | One of the descriptor sets specified an invalid descriptor. |
| EINTR | A signal was delivered before any of the selected events occurred, or before the time limit expired. |
| EINVAL | A component of the pointed-to time limit is outside the acceptable range: **t_sec** must be between 0 and $10^8$, inclusive. **t_usec** must be greater-than or equal to 0, and less than $10^6$. |
| EFAULT | One of the pointers given in the call referred to a non-existent portion of the process' address space. |

**SEE ALSO**

**accept**(2), **connect**(2), **gettimeofday**(2), **read**(2V), **write**(2V), **recv**(2), **send**(2), **getdtablesize**(2)

**BUGS**

Although the provision of **getdtablesize**(2) was intended to allow user programs to be written independent of the kernel limit on the number of open files, the dimension of a sufficiently large bit field for select remains a problem. The default size **FD_SETSIZE** (currently 256) is somewhat larger than the current kernel limit to the number of open files. However, in order to accommodate programs which might potentially use a larger number of open files with select, it is possible to increase this size within a program by providing a larger definition of **FD_SETSIZE** before the inclusion of **<sys/types.h>**.

**select**( ) should probably return the time remaining from the original timeout, if any, by modifying the time value in place. This may be implemented in future versions of the system. Thus, it is unwise to assume that the timeout pointer will be unmodified by the **select**( ) call.

**NAME**

semctl – semaphore control operations

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl (semid, semnum, cmd, arg)
int semid, cmd;
int semnum;
union semun {
        val;
        struct semid_ds *buf;
        ushort *array;
} arg;
```

**DESCRIPTION**

**semctl( )** provides a variety of semaphore control operations as specified by *cmd*.

The following *cmd*s are executed with respect to the semaphore specified by *semid* and *semnum:*

| | |
|---|---|
| **GETVAL** | Return the value of *semval* (see **intro**(2)). {READ} |
| **SETVAL** | Set the value of *semval* to *arg.val*. {ALTER} When this cmd is successfully executed, the *semadj* value corresponding to the specified semaphore in all processes is cleared. |
| **GETPID** | Return the value of *sempid*. {READ} |
| **GETNCNT** | Return the value of *semncnt*. {READ} |
| **GETZCNT** | Return the value of *semzcnt*. {READ} |

The following *cmd*s return and set, respectively, every *semval* in the set of semaphores.

**GETALL**
Place *semvals* into the array pointed to by *arg.array*. {READ}

**SETALL**
Set *semvals* according to the array pointed to by *arg.array*. {ALTER} When this cmd is successfully executed the *semadj* values corresponding to each specified semaphore in all processes are cleared.

The following *cmd*s are also available:

**IPC_STAT** Place the current value of each member of the data structure associated with *semid* into the structure pointed to by *arg.buf*. The contents of this structure are defined in **intro**(2). {READ}

**IPC_SET** Set the value of the following members of the data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*:

**sem_perm.uid**
**sem_perm.gid**
**sem_perm.mode /* only low 9 bits */**

This *cmd* can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **sem_perm.cuid** or **sem_perm.uid** in the data structure associated with *semid*.

**IPC_RMID** Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and data structure associated with it. This cmd can only be executed by a process that has an effective user ID equal to either that of super-user, or to the value of **sem_perm.cuid** or **sem_perm.uid** in the data structure associated with *semid*.

## RETURN VALUE

Upon successful completion, the value returned depends on *cmd* as follows:

| | |
|---|---|
| **GETVAL** | The value of *semval*. |
| **GETPID** | The value of *sempid*. |
| **GETNCNT** | The value of *semncnt*. |
| **GETZCNT** | The value of *semzcnt*. |
| All others | A value of 0. |

Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**semctl**( ) will fail if one or more of the following are true:

| | |
|---|---|
| EINVAL | *semid* is not a valid semaphore identifier. |
| EINVAL | *semnum* is less than zero or greater than **sem_nsems**. |
| EINVAL | *cmd* is not a valid command. |
| EACCES | Operation permission is denied to the calling process (see **intro**(2)). |
| ERANGE | *cmd* is **SETVAL** or **SETALL** and the value to which *semval* is to be set is greater than the system imposed maximum. |
| EPERM | *cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not equal to that of super-user, or to the value of **sem_perm.cuid** or **sem_perm.uid** in the data structure associated with *semid*. |
| EFAULT | *arg.buf* points to an illegal address. |

## SEE ALSO

**intro**(2), **semget**(2), **semop**(2).

## NAME

semget – get set of semaphores

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(key, nsems, semflg)
key_t key;
int nsems, semflg;
```

## DESCRIPTION

**semget()** returns the semaphore identifier associated with *key*.

A semaphore identifier and associated data structure and set containing *nsems* semaphores (see **intro**(2)) are created for *key* if one of the following are true:

- *key* is equal to **IPC_PRIVATE**.
- *key* does not already have a semaphore identifier associated with it, and (*semflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

- **sem_perm.cuid**, **sem_perm.uid**, **sem_perm.cgid**, and **sem_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **sem_perm.mode** are set equal to the low-order 9 bits of *semflg*.
- **sem_nsems** is set equal to the value of *nsems*.
- **sem_otime** is set equal to 0 and **sem_ctime** is set equal to the current time.

## RETURN VALUE

Upon successful completion, a non-negative integer, namely a semaphore identifier, is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**semget()** will fail if one or more of the following are true:

| | |
|---|---|
| EINVAL | *nsems* is either less than or equal to zero or greater than the system-imposed limit. |
| EACCES | A semaphore identifier exists for *key*, but operation permission (see **intro**(2)) as specified by the low-order 9 bits of *semflg* would not be granted. |
| EINVAL | A semaphore identifier exists for *key*, but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to zero. |
| ENOENT | A semaphore identifier does not exist for *key* and (*semflg* & **IPC_CREAT**) is "false". |
| ENOSPC | A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded. |
| ENOSPC | A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system wide would be exceeded. |
| EEXIST | A semaphore identifier exists for *key* but ( (*semflg* & **IPC_CREAT**) and (*semflg* & **IPC_EXCL**)) is "true". |

## SEE ALSO

**intro**(2), **semctl**(2), **semop**(2)

NAME

semop – semaphore operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semop(semid, sops, nsops)
int semid;
struct sembuf *sops;
int nsops;
```

DESCRIPTION

**semop()** is used to atomically perform an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by *semid*. *sops* is a pointer to the array of semaphore-operation structures. *nsops* is the number of such structures in the array. The contents of each structure includes the following members:

```
short   sem_num;   /* semaphore number */
short   sem_op;    /* semaphore operation */
short   sem_flg;   /* operation flags */
```

Each semaphore operation specified by **sem_op** is performed on the corresponding semaphore specified by *semid* and **sem_num**.

**sem_op** specifies one of three semaphore operations as follows:

If **sem_op** is a negative integer, one of the following will occur: {ALTER}

- If *semval* (see **intro**(2)) is greater than or equal to the absolute value of **sem_op**, the absolute value of **sem_op** is subtracted from *semval*. Also, if (**sem_flg** & SEM_UNDO) is "true", the absolute value of **sem_op** is added to the calling process's *semadj* value (see **exit**(2)) for the specified semaphore.
- If *semval* is less than the absolute value of **sem_op** and (**sem_flg** & IPC_NOWAIT) is "true", **semop()** will return immediately.
- If *semval* is less than the absolute value of **sem_op** and (**sem_flg** & IPC_NOWAIT) is "false", **semop()** will increment the *semncnt* associated with the specified semaphore and suspend execution of the calling process until one of the following conditions occur.

  *semval* becomes greater than or equal to the absolute value of **sem_op**. When this occurs, the value of *semncnt* associated with the specified semaphore is decremented, the absolute value of *sem_op* is subtracted from *semval* and, if (**sem_flg** & SEM_UNDO) is "true", the absolute value of **sem_op** is added to the calling process's *semadj* value for the specified semaphore.

  The *semid* for which the calling process is awaiting action is removed from the system (see **semctl**(2)). When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.

  The calling process receives a signal that is to be caught. When this occurs, the value of *semncnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in **signal**(2).

If **sem_op** is a positive integer, the value of **sem_op** is added to *semval* and, if (**sem_flg** & SEM_UNDO) is "true", the value of **sem_op** is subtracted from the calling process's *semadj* value for the specified semaphore. {ALTER}

If **sem_op** is zero, one of the following will occur: {READ}

- If *semval* is zero, **semop()** will return immediately.
- If *semval* is not equal to zero and (**sem_flg** & IPC_NOWAIT) is "true", **semop()** will return immediately.
- If *semval* is not equal to zero and (**sem_flg** & IPC_NOWAIT) is "false", **semop()** will increment the *semzcnt* associated with the specified semaphore and suspend execution of the calling process until one of the following occurs:

  *semval* becomes zero, at which time the value of *semzcnt* associated with the specified semaphore is decremented.

  The *semid* for which the calling process is awaiting action is removed from the system. When this occurs, **errno** is set equal to EIDRM, and a value of −1 is returned.

  The calling process receives a signal that is to be caught. When this occurs, the value of *semzcnt* associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in **signal**(2).

Upon successful completion, the value of *sempid* for each semaphore specified in the array pointed to by *sops* is set equal to the process ID of the calling process.

## RETURN VALUE

Upon successful completion, the value of *semval* at the time of the call for the last operation in the array pointed to by *sops* is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**semop()** will fail if one or more of the following are true for any of the semaphore operations specified by *sops*:

| | |
|---|---|
| EINVAL | *semid* is not a valid semaphore identifier. |
| EIDRM | The set of semaphores referred to by *msqid* was removed from the system. |
| EFBIG | **sem_num** is less than zero or greater than or equal to the number of semaphores in the set associated with *semid*. |
| E2BIG | *nsops* is greater than the system-imposed maximum. |
| EACCES | Operation permission is denied to the calling process (see **intro**(2)). |
| EAGAIN | The operation would result in suspension of the calling process but (**sem_flg** & IPC_NOWAIT) is "true". |
| ENOSPC | The limit on the number of individual processes requesting an SEM_UNDO would be exceeded. |
| EINVAL | The number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the limit. |
| ERANGE | An operation would cause a *semval* or *semadj* value to overflow the system-imposed limit. |
| EFAULT | *sops* points to an illegal address. |
| EINTR | The call was interrupted by the delivery of a signal. |

## SEE ALSO

**exec**(2), **exit**(2), **fork**(2), **intro**(2), **semctl**(2), **semget**(2), **signal**(2)

NAME

send, sendto, sendmsg – send a message from a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int send(s, msg, len, flags)
int s;
char *msg;
int len, flags;

int sendto(s, msg, len, flags, to, tolen)
int s;
char *msg;
int len, flags;
struct sockaddr *to;
int tolen;

int sendmsg(s, msg, flags)
int s;
struct msghdr *msg;
int flags;
```

DESCRIPTION

*s* is a socket created with **socket**(2). **send**, **sendto**, and **sendmsg**() are used to transmit a message to another socket. **send**() may be used only when the socket is in a *connected* state, while **sendto**() and **sendmsg**() may be used at any time.

The address of the target is given by *to* with *tolen* specifying its size. The length of the message is given by *len*. If the message is too long to pass atomically through the underlying protocol, then the error EMSGSIZE is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a **send**. Return values of −1 indicate some locally detected errors.

If no buffer space is available at the socket to hold the message to be transmitted, then **send**() normally blocks, unless the socket has been placed in non-blocking I/O mode. The **select**(2) call may be used to determine when it is possible to send more data.

The *flags* parameter is formed by ORing one or more of the following:

MSG_OOB
: Send "out-of-band" data on sockets that support this notion. The underlying protocol must also support "out-of-band" data. Currently, only SOCK_STREAM sockets created in the AF_INET address family support out-of-band data.

MSG_DONTROUTE
: The SO_DONTROUTE option is turned on for the duration of the operation. This is usually used only by diagnostic or routing programs.

See **recv**(2) for a description of the **msghdr** structure.

RETURN VALUE

These calls return the number of bytes sent, or −1 if an error occurred.

ERRORS

The calls fail if:

| | |
|---|---|
| EBADF | *s* is an invalid descriptor. |
| ENOTSOCK | *s* is a descriptor for a file, not a socket. |
| EINVAL | *len* is not the size of a valid address for the specified address family. |

| | |
|---|---|
| EINTR | The operation was interrupted by delivery of a signal before any data could be buffered to be sent. |
| EFAULT | The data was specified to be sent to a non-existent or protected part of the process address space. |
| EMSGSIZE | The socket requires that message be sent atomically, and the size of the message to be sent made this impossible. |
| EWOULDBLOCK | The socket is marked non-blocking and the requested operation would block. |
| ENOBUFS | The system was unable to allocate an internal buffer. The operation may succeed when buffers become available. |
| ENOBUFS | The output queue for a network interface was full. This generally indicates that the interface has stopped sending, but may be caused by transient congestion. |

**SEE ALSO**

**connect**(2), **fcntl**(2V), **getsockopt**(2), **recv**(2), **select**(2), **socket**(2), **write**(2V)

## NAME

setpgrp, getpgrp – set and/or return the process group of a process

## SYNOPSIS

```
int setpgrp(pid, pgrp)
int pgrp;
int pid;

int getpgrp(pid)
int pid;
```

## SYSTEM V SYNOPSIS

```
int setpgrp ()

int getpgrp ()
```

## DESCRIPTION

### setpgrp

**setpgrp**( ) sets the process group of the specified process, (*pid*) to the process group specified by *pgrp*. If *pid* is zero, then the call applies to the current (calling) process.

If the effective user ID is not that of the super-user, then the process to be affected must have the same effective user ID as that of the caller or be a descendant of that process.

### getpgrp

**getpgrp**( ) returns the process group of the indicated process. If *pid* is zero, then the call applies to the calling process.

Process groups are used for distribution of signals, and by terminals to arbitrate requests for their input. Processes that have the same process group as the terminal run in the foreground and may read from the terminal, while others block with a signal when they attempt to read.

This call is thus used by programs such as **csh**(1) to create process groups in implementing job control. The **TIOCGPGRP** and **TIOCSPGRP** calls described in **termio**(4) are used to get/set the process group of the control terminal.

## SYSTEM V DESCRIPTION

### setpgrp

**setpgrp**( ) sets the process group of the calling process to match its process ID, and returns the new process group ID.

### getpgrp

**getpgrp**( ) returns the process group of the calling process.

## RETURN VALUE

**setpgrp**( ) returns 0 when the operation was successful. If the request failed, –1 is returned and the global variable **errno** indicates the reason.

## ERRORS

**setpgrp**( ) fails, and the process group is not altered when one of the following occurs:

| | |
|---|---|
| ESRCH | The requested process does not exist. |
| EPERM | The effective user ID of the requested process is different from that of the caller and the process is not a descendent of the calling process. |

## SEE ALSO

**csh**(1), **execve**(2), **fork**(2), **getpid**(2), **getuid**(2), **intro**(2), **kill**(2V), **signal**(3), **termio**(4)

**NAME**

setregid – set real and effective group IDs

**SYNOPSIS**

**int setregid(rgid, egid)**
**int rgid, egid;**

**DESCRIPTION**

**setregid( )** is used to set the real and effective group IDs of the calling process. If *rgid* is −1, the real GID is not changed; if *egid* is −1, the effective GID is not changed. The real and effective GIDs may be set to different values in the same call.

If the effective user ID of the calling process is super-user, the real GID and the effective GID can be set to any legal value.

If the effective user ID of the calling process is not super-user, either the real GID can be set to the saved setGID from **execve**(2), or the effective GID can either be set to the saved setGID or the real GID. Note: if a setGID process sets its effective GID to its real GID, it can still set its effective GID back to the saved setGID.

In either case, if the real GID is being changed (that is, if *rgid* is not −1), or the effective GID is being changed to a value not equal to the real GID, the saved setGID is set equal to the new effective GID.

If the real GID is changed from its current value, the old value is removed from the groups access list (see **getgroups**(2)) if it is present in that list, and the new value is added to the groups access list if it is not already present and if this would not cause the number of groups in that list to exceed **NGROUPS**, as defined in **<sys/param.h>**.

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

**ERRORS**

**setregid( )** will fail and neither of the group IDs will be changed if:

EPERM — The calling process' effective UID is not the super-user and a change other than changing the real GID to the saved setGID, or changing the effective GID to the real GID or the saved GID, was specified.

**SEE ALSO**

**getgid**(2), **execve**(2), **setreuid**(2), **setuid**(3)

**NAME**

setreuid – set real and effective user IDs

**SYNOPSIS**

**int setreuid(ruid, euid)**
**int ruid, euid;**

**DESCRIPTION**

**setreuid( )** is used to set the real and effective user IDs of the calling process. If *ruid* is −1, the real user ID is not changed; if *euid* is −1, the effective user ID is not changed. The real and effective user IDs may be set to different values in the same call.

If the effective user ID of the calling process is super-user, the real user ID and the effective user ID can be set to any legal value.

If the effective user ID of the calling process is not super-user, either the real user ID can be set to the effective user ID, or the effective user ID can either be set to the saved set-user ID from **execve**(2) or the real user ID. Note: if a set-UID process sets its effective user ID to its real user ID, it can still set its effective user ID back to the saved set-user ID.

In either case, if the real user ID is being changed (that is, if *ruid* is not −1), or the effective user ID is being changed to a value not equal to the real user ID, the saved set-user ID is set equal to the new effective user ID.

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

**ERRORS**

**setreuid( )** will fail and neither of the user IDs will be changed if:

| | |
|---|---|
| EPERM | The calling process' effective user ID is not the super-user and a change other than changing the real user ID to the effective user ID, or changing the effective user ID to the real user ID or the saved set-user ID, was specified. |

**SEE ALSO**

**execve**(2), **getuid**(2), **setregid**(2), **setuid**(3)

**NAME**

setuseraudit, setaudit – set the audit classes for a specified user ID

**SYNOPSIS**

```
#include <sys/label.h>
#include <sys/audit.h>

setuseraudit(uid, state)
int uid;
audit_state_t *state;

setaudit(state)
audit_state_t *state;
```

**DESCRIPTION**

The **setuseraudit()** system call sets the audit state for all processes whose audit user ID matches the specified user ID. The parameter *state* specifies the audit classes to audit for both successful and unsuccessful operations.

The **setaudit** system call sets the audit state for the current process.

Only processes with the real or effective user ID of the super-user may successfully execute these calls.

**RETURN VALUE**

If the call succeeds, a value of 0 is returned. If an error occurs, the value −1 is returned.

**ERRORS**

| | |
|---|---|
| EPERM | The process' real or effective user ID is not super-user. |
| EFAULT | The *state* parameter points outside the processes' allocated address space. |

**SEE ALSO**

**audit**(2), **audit_args**(3), **audit_control**(5), **audit.log**(5)

## NAME

shmctl – shared memory control operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmctl (shmid, cmd, buf)
int shmid, cmd;
struct shmid_ds *buf;
```

## DESCRIPTION

**shmctl**( ) provides a variety of shared memory control operations as specified by *cmd*. The following *cmd*s are available:

IPC_STAT — Place the current value of each member of the data structure associated with *shmid* into the structure pointed to by *buf*. The contents of this structure are defined in **intro**(2). {READ}

IPC_SET — Set the value of the following members of the data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

**shm_perm.uid**
**shm_perm.gid**
**shm_perm.mode /* only low 9 bits */**

This *cmd* can only be executed by a process that has an effective user ID equal to that of super-user, or to the value of **shm_perm.cuid** or **shm_perm.uid** in the data structure associated with *shmid*.

IPC_RMID — Remove the shared memory identifier specified by *shmid* from the system. If no processes are currently mapped to the corresponding shared memory segment, then the segment is removed and the associated resources are reclaimed. Otherwise, the segment will persist, although **shmget**(2) will not be able to locate it, until it is no longer mapped by any process. This *cmd* can only be executed by a process that has an effective user ID equal to that of super-user, or to the value of **shm_perm.cuid** or **shm_perm.uid** in the data structure associated with *shmid*.

## RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**shmctl**( ) will fail if one or more of the following are true:

EINVAL — *shmid* is not a valid shared memory identifier.

EINVAL — *cmd* is not a valid command.

EACCES — *cmd* is equal to **IPC_STAT** and {READ} operation permission is denied to the calling process (see **intro**(2)).

EPERM — *cmd* is equal to **IPC_RMID** or **IPC_SET** and the effective user ID of the calling process is not equal to that of super-user, or to the value of **shm_perm.cuid** or **shm_perm.uid** in the data structure associated with *shmid*.

EFAULT — *buf* points to an illegal address.

## SEE ALSO

**intro**(2), **shmget**(2), **shmop**(2)

## NAME

shmget – get shared memory segment identifier

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int shmget(key, size, shmflg)
key_t key;
int size, shmflg;
```

## DESCRIPTION

**shmget( )** returns the shared memory identifier associated with *key*.

A shared memory identifier and associated data structure and shared memory segment of at least *size* bytes (see **intro**(2)) are created for *key* if one of the following are true:

- *key* is equal to **IPC_PRIVATE.**
- *key* does not already have a shared memory identifier associated with it, and (*shmflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:

- **shm_perm.cuid, shm_perm.uid, shm_perm.cgid,** and **shm_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.
- The low-order 9 bits of **shm_perm.mode** are set equal to the low-order 9 bits of *shmflg*.
- **shm_segsz** is set equal to the value of *size*.
- **shm_lpid, shm_nattch, shm_atime,** and **shm_dtime** are set equal to 0.
- **shm_ctime** is set equal to the current time.

## RETURN VALUE

Upon successful completion, a non-negative integer, namely a shared memory identifier, is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**shmget( )** will fail if one or more of the following are true:

| | |
|---|---|
| EINVAL | *size* is less than the system-imposed minimum or greater than the system-imposed maximum. |
| EACCES | A shared memory identifier exists for *key* but operation permission (see **intro**(2)) as specified by the low-order 9 bits of *shmflg* would not be granted. |
| EINVAL | A shared memory identifier exists for *key* but the size of the segment associated with it is less than *size* and *size* is not equal to zero. |
| ENOENT | A shared memory identifier does not exist for *key* and (*shmflg* & **IPC_CREAT**) is "false". |
| ENOSPC | A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory identifiers system wide would be exceeded. |
| ENOMEM | A shared memory identifier and associated shared memory segment are to be created but the amount of available physical memory is not sufficient to fill the request. |
| EEXIST | A shared memory identifier exists for *key* but ( (*shmflg* & **IPC_CREAT**) and (*shmflg* & **IPC_EXCL**) ) is "true". |

**SEE ALSO**

**intro**(2), **shmctl**(2), **shmop**(2)

## NAME

shmop, shmat, shmdt – shared memory operations

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

char *shmat(shmid, shmaddr, shmflg)
int shmid;
char *shmaddr;
int shmflg;

int shmdt(shmaddr)
char *shmaddr;
```

## DESCRIPTION

**shmat()** maps the shared memory segment associated with the shared memory identifier specified by *shmid* into the data segment of the calling process. Upon successful completion, the address of the mapped segment is returned.

The shared memory segment is mapped at the address specified by one of the following criteria:

- If *shmaddr* is equal to zero, the segment is mapped at an address selected by the system. Ordinarily, applications should invoke **shmat()** with *shmaddr* equal to zero so that the operating system may make the best use of available resources.
- If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is "true", the segment is mapped at the address given by (*shmaddr* - (*shmaddr* modulus **SHMLBA**)).
- If *shmaddr* is not equal to zero and (*shmflg* & **SHM_RND**) is "false", the segment is mapped at the address given by *shmaddr*.

The segment is mapped for reading if (*shmflg* & **SHM_RDONLY**) is "true" **{READ}**, otherwise it is mapped for reading and writing **{READ/WRITE}**.

**shmdt()** unmaps from the calling process's address space the shared memory segment that is mapped at the address specified by *shmaddr*. The shared memory segment must have been mapped with a prior **shmat()** function call. The segment and contents are retained until explicitly removed by means of the **IPC_RMID** function (see **shmctl**(2)).

## RETURN VALUES

Upon successful completion, the return values are as follows:

- **shmat()** returns the data segment start address of the mapped shared memory segment.
- **shmdt()** returns a value of 0.

Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**shmat()** will fail and not map the shared memory segment if one or more of the following are true:

| | |
|---|---|
| EINVAL | *shmid* is not a valid shared memory identifier. |
| EACCES | Operation permission is denied to the calling process (see **intro**(2)). |
| ENOMEM | The available data space is not large enough to accommodate the shared memory segment. |
| EINVAL | *shmaddr* is not equal to zero, and the value of (*shmaddr* - (*shmaddr* modulus **SHMLBA**)) is an illegal address. |
| EINVAL | *shmaddr* is not equal to zero, (*shmflg* & **SHM_RND**) is "false", and the value of *shmaddr* is an illegal address. |

EMFILE The number of shared memory segments mapped to the calling process would exceed the system-imposed limit.

**shmdt**( ) will fail and not unmap the shared memory segment if:

EINVAL
*shmaddr* is not the data segment start address of a shared memory segment.

SEE ALSO
**execve**(2), **exit**(2), **fork**(2), **intro**(2), **shmctl**(2), **shmget**(2)

**NAME**

shutdown – shut down part of a full-duplex connection

**SYNOPSIS**

**shutdown(s, how)**
**int s, how;**

**DESCRIPTION**

The **shutdown( )** call causes all or part of a full-duplex connection on the socket associated with *s* to be shut down. If *how* is 0, then further receives will be disallowed. If *how* is 1, then further sends will be disallowed. If *how* is 2, then further sends and receives will be disallowed.

**DIAGNOSTICS**

A 0 is returned if the call succeeds, –1 if it fails.

**ERRORS**

The call succeeds unless:

| | |
|---|---|
| EBADF | *s* is not a valid descriptor. |
| ENOTSOCK | *s* is a file, not a socket. |
| ENOTCONN | The specified socket is not connected. |

**SEE ALSO**

**connect**(2), **socket**(2)

**BUGS**

The *how* values should be defined constants.

## NAME

sigblock, sigmask – block signals

## SYNOPSIS

```
#include <signal.h>

oldmask = sigblock(mask);
int mask;

mask = sigmask(signum)
```

## DESCRIPTION

**sigblock()** adds the signals specified in *mask* to the set of signals currently being blocked from delivery. Signals are blocked if the appropriate bit in *mask* is a 1; the macro **sigmask()** is provided to construct the mask for a given *signum*. The previous mask is returned, and may be restored using **sigsetmask**(2).

It is not possible to block **SIGKILL**, **SIGSTOP**, or **SIGCONT**; this restriction is silently imposed by the system.

## RETURN VALUE

The previous set of masked signals is returned.

## SEE ALSO

**kill**(2V), **sigsetmask**(2), **sigvec**(2), **signal**(3)

NAME

sigpause – atomically release blocked signals and wait for interrupt

SYNOPSIS

**sigpause(sigmask)**

**int sigmask;**

DESCRIPTION

**sigpause()** assigns *sigmask* to the set of masked signals and then waits for a signal to arrive; on return the set of masked signals is restored. *sigmask* is usually 0 to indicate that no signals are now to be blocked. **sigpause()** always terminates by being interrupted, returning EINTR.

In normal usage, a signal is blocked using **sigblock**(2), to begin a critical section, variables modified on the occurrence of the signal are examined to determine that there is no work to be done, and the process pauses awaiting work by using **sigpause()** with the mask returned by *sigblock*.

SEE ALSO

**sigblock**(2), **sigvec**(2), **signal**(3)

## NAME

sigsetmask – set current signal mask

## SYNOPSIS

```
#include <signal.h>

sigsetmask(mask);
int mask;

mask = sigmask(signum)
```

## DESCRIPTION

**sigsetmask()** sets the current signal mask (those signals that are blocked from delivery). Signals are blocked if the corresponding bit in *mask* is a 1; the macro *sigmask* is provided to construct the mask for a given *signum*.

The system quietly disallows **SIGKILL**, **SIGSTOP**, or **SIGCONT** from being blocked.

## RETURN VALUE

The previous set of masked signals is returned.

## SEE ALSO

**kill**(2V), **sigblock**(2), **sigpause**(2), **sigvec**(2), **signal**(3)

NAME

sigstack – set and/or get signal stack context

SYNOPSIS

```
#include <signal.h>

int sigstack (ss, oss)
struct sigstack *ss, *oss;
```

DESCRIPTION

**sigstack()** allows users to define an alternate stack, called the "signal stack", on which signals are to be processed. When a signal's action indicates its handler should execute on the signal stack (specified with a *sigvec*(2) call), the system checks to see if the process is currently executing on that stack. If the process is not currently executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler's execution.

A signal stack is specified by a **sigstack()** structure, which includes the following members:

```
char    *ss_sp;     /* signal stack pointer */
int     ss_onstack; /* current status */
```

**ss_sp** is the initial value to be assigned to the stack pointer when the system switches the process to the signal stack. Note that, on machines where the stack grows downwards in memory, this is *not* the address of the beginning of the signal stack area. **ss_onstack** field is zero or non-zero depending on whether the process is currently executing on the signal stack or not.

If *ss* is not a NULL pointer, **sigstack()** sets the signal stack state to the value in the **sigstack()** structure pointed to by *ss*. Note: if **ss_onstack** is non-zero, the system will think that the process is executing on the signal stack. If *ss* is a NULL pointer, the signal stack state will be unchanged. If *oss* is not a NULL pointer, the current signal stack state is stored in the **sigstack()** structure pointed to by *oss*.

NOTES

Signal stacks are not "grown" automatically, as is done for the normal stack. If the stack overflows unpredictable results may occur.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

ERRORS

**sigstack()** will fail and the signal stack context will remain unchanged if one of the following occurs.

| | |
|---|---|
| EFAULT | Either *ss* or *oss* points to memory that is not a valid part of the process address space. |

SEE ALSO

**sigvec**(2), **setjmp**(3), **signal**(3)

NAME

sigvec – software signal facilities

SYNOPSIS

**#include <signal.h>**

**int sigvec(sig, vec, ovec)**
**int sig;**
**struct sigvec *vec, *ovec;**

DESCRIPTION

The system defines a set of signals that may be delivered to a process. Signal delivery resembles the occurrence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a *handler* to which a signal is delivered, or specify that a signal is to be *blocked* or *ignored*. A process may also specify that a default action is to be taken by the system when a signal occurs. Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special *signal stack*.

All signals have the same *priority*. Signal routines execute with the signal that caused their invocation *blocked*, but other signals may yet occur. A global *signal mask* defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a **sigblock**(2) or **sigsetmask**(2) call, or when a signal is delivered to the process.

A process may also specify a set of *flags* for a signal that affect the delivery of that signal.

When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently *blocked* by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally the process will resume execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself.

When a signal is delivered to a process a new signal mask is installed for the duration of the process' signal handler (or until a **sigblock**( ) or **sigsetmask**( ) call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and ORing in the signal mask associated with the handler to be invoked.

The action to be taken when the signal is delivered is specified by a **sigvec**( ) structure, which includes the following members:

```
void    (*sv_handler)();    /* signal handler */
int     sv_mask;            /* signal mask to apply */
int     sv_flags;           /* see signal options */

#define SV_ONSTACK      0x0001      /* take signal on signal stack */
#define SV_INTERRUPT    0x0002      /* do not restart system on signal return */
#define SV_RESETHAND    0x0004      /* reset signal handler to SIG_DFL when signal taken */
```

If the SV_ONSTACK bit is set in the flags for that signal, the system will deliver the signal to the process on the signal stack specified with **sigstack**(2), rather than delivering the signal on the current stack.

If *vec* is not a NULL pointer, **sigvec**( ) assigns the handler specified by **sv_handler**, the mask specified by **sv_mask**, and the flags specified by **sv_flags** to the specified signal. If *vec* is a NULL pointer, **sigvec**( ) does not change the handler, mask, or flags for the specified signal.

The mask specified in *vec* is not allowed to block **SIGKILL**, **SIGSTOP**, or **SIGCONT**. The system enforces this restriction silently.

If *ovec* is not a NULL pointer, the handler, mask, and flags in effect for the signal before the call to **sigvec**( ) are returned to the user. A call to **sigvec**( ) with *vec* a NULL pointer and *ovec* not a NULL pointer can be used to determine the handling information currently in effect for a signal without changing that

information.

The following is a list of all signals with names as in the include file **<signal.h>**:

| | | |
|---|---|---|
| **SIGHUP** | 1 | hangup |
| **SIGINT** | 2 | interrupt |
| **SIGQUIT** | 3* | quit |
| **SIGILL** | 4* | illegal instruction |
| **SIGTRAP** | 5* | trace trap |
| **SIGABRT** | 6* | abort (generated by **abort**(3) routine) |
| **SIGEMT** | 7* | emulator trap |
| **SIGFPE** | 8* | arithmetic exception |
| **SIGKILL** | 9 | kill (cannot be caught, blocked, or ignored) |
| **SIGBUS** | 10* | bus error |
| **SIGSEGV** | 11* | segmentation violation |
| **SIGSYS** | 12* | bad argument to system call |
| **SIGPIPE** | 13 | write on a pipe or other socket with no one to read it |
| **SIGALRM** | 14 | alarm clock |
| **SIGTERM** | 15 | software termination signal |
| **SIGURG** | 16• | urgent condition present on socket |
| **SIGSTOP** | 17† | stop (cannot be caught, blocked, or ignored) |
| **SIGTSTP** | 18† | stop signal generated from keyboard |
| **SIGCONT** | 19• | continue after stop (cannot be blocked) |
| **SIGCHLD** | 20• | child status has changed |
| **SIGTTIN** | 21† | background read attempted from control terminal |
| **SIGTTOU** | 22† | background write attempted to control terminal |
| **SIGIO** | 23• | I/O is possible on a descriptor (see **fcntl**(2V)) |
| **SIGXCPU** | 24 | cpu time limit exceeded (see **setrlimit**(2)) |
| **SIGXFSZ** | 25 | file size limit exceeded (see **setrlimit**(2)) |
| **SIGVTALRM** | 26 | virtual time alarm (see **setitimer**(2)) |
| **SIGPROF** | 27 | profiling timer alarm (see **setitimer**(2)) |
| **SIGWINCH** | 28• | window changed (see **termio**(4) and **win**(4S)) |
| **SIGLOST** | 29* | resource lost (see **lockd**(8C)) |
| **SIGUSR1** | 30 | user-defined signal 1 |
| **SIGUSR2** | 31 | user-defined signal 2 |

The starred signals in the list above cause a core image if not caught or ignored.

Once a signal handler is installed, it remains installed until another **sigvec( )** call is made, or an **execve**(2) is performed, unless the **SV_RESETHAND** bit is set in the flags for that signal. In that case, the value of the handler for the caught signal will be set to **SIG_DFL** before entering the signal-catching function, unless the signal is **SIGILL** or **SIGTRAP**. Also, if this bit is set, the bit for that signal in the signal mask will not be set; unless the signal mask associated with that signal blocks that signal, further occurrences of that signal will not be blocked. The **SV_RESETHAND** flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

The default action for a signal may be reinstated by setting the signal's handler to **SIG_DFL**; this default is termination except for signals marked with • or †. Signals marked with • are discarded if the action is **SIG_DFL**; signals marked with † cause the process to stop. If the process is terminated, a "core image" will be made in the current working directory of the receiving process if the signal is one for which an asterisk appears in the above list *and* the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

The effective group ID and the real group ID of the receiving process are equal.

An ordinary file named **core** exists and is writable or can be created. If the file must be created, it will have the following properties:

- a mode of 0666 modified by the file creation mask (see **umask**(2))
- a file owner ID that is the same as the effective user ID of the receiving process.
- a file group ID that is the same as the file group ID of the current directory

If the handler for that signal is SIG_IGN, the signal is subsequently ignored, and pending instances of the signal are discarded.

Note: the signals SIGKILL, SIGSTOP, and SIGCONT cannot be ignored.

If a caught signal occurs during certain system calls, the call is normally restarted. The call can be forced to terminate prematurely with an EINTR error return by setting the SV_INTERRUPT bit in the flags for that signal. The SV_INTERRUPT flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed. The affected system calls are **read**(2V) or **write**(2V) on a slow device (such as a terminal or pipe or other socket, but not a file) and during a **wait**(2).

After a **fork**(2) or **vfork**(2) the child inherits all signals, the signal mask, the signal stack, and the restart/interrupt and reset-signal-handler flags.

The **execve**(2) call resets all caught signals to default action and resets all signals to be caught on the user stack. Ignored signals remain ignored; the signal mask remains the same; signals that interrupt system calls continue to do so.

## NOTES

SIGPOLL is a synonym for SIGIO. A SIGIO will be issued when a file descriptor corresponding to a STREAMS (see **intro**(2)) file has a "selectable" event pending. Unless that descriptor has been put into asynchronous mode (see **fcntl (2V)**, a process must specifically request that this signal be sent using the I_SETSIG **ioctl**(2) call (see **streamio**(4)). Otherwise, the process will never receive SIGPOLL.

The handler routine can be declared:

```
void handler(sig, code, scp, addr)
int sig, code;
struct sigcontext *scp;
char *addr;
```

Here *sig* is the signal number; *code* is a parameter of certain signals that provides additional detail; *scp* is a pointer to the **sigcontext** structure (defined in **<signal.h>**), used to restore the context from before the signal; and *addr* is additional address information.

Programs that must be portable to UNIX systems other than 4.2BSD should use the **signal**(3) interface instead.

## CODES

The following defines the codes for signals which produce them. All of these symbols are defined in **<signal.h>**:

| Condition | Signal | Code |
|---|---|---|
| Sun codes: | | |
| Illegal instruction | **SIGILL** | **ILL_INSTR_FAULT** |
| Integer division by zero | **SIGFPE** | **FPE_INTDIV_TRAP** |
| IEEE floating pt inexact | **SIGFPE** | **FPE_FLTINEX_TRAP** |
| IEEE floating pt division by zero | **SIGFPE** | **FPE_FLTDIV_TRAP** |
| IEEE floating pt underflow | **SIGFPE** | **FPE_FLTUND_TRAP** |
| IEEE floating pt operand error | **SIGFPE** | **FPE_FLTOPERR_TRAP** |
| IEEE floating pt overflow | **SIGFPE** | **FPE_FLTOVF_FAULT** |
| Hardware bus error | **SIGBUS** | **BUS_HWERR** |
| Address alignment error | **SIGBUS** | **BUS_ALIGN** |
| No mapping fault | **SIGSEGV** | **SEGV_NOMAP** |

| | | |
|---|---|---|
| Protection fault | **SIGSEGV** | **SEGV_PROT** |
| Object error | **SIGSEGV** | **SEGV_CODE(code)=SEGV_OBJERR** |
| Object error number | **SIGSEGV** | **SEGV_ERRNO(code)** |
| SPARC codes: | | |
| Privileged instruction violation | **SIGILL** | **ILL_PRIVINSTR_FAULT** |
| Bad stack | **SIGILL** | **ILL_STACK** |
| Trap *#n* (1 <= *n* <= 127) | **SIGILL** | **ILL_TRAP_FAULT(*n*)** |
| Tag overflow | **SIGEMT** | **EMT_TAG** |
| MC680X0 codes: | | |
| Privilege violation | **SIGILL** | **ILL_PRIVVIO_FAULT** |
| Coprocessor protocol error | **SIGILL** | **ILL_INSTR_FAULT** |
| Trap *#n* (1 <= *n* <= 14) | **SIGILL** | **ILL_TRAP*n*_FAULT** |
| A-line op code | **SIGEMT** | **EMT_EMU1010** |
| F-line op code | **SIGEMT** | **EMT_EMU1111** |
| CHK or CHK2 instruction | **SIGFPE** | **FPE_CHKINST_TRAP** |
| TRAPV or TRAPcc or cpTRAPcc | **SIGFPE** | **FPE_TRAPV_TRAP** |
| IEEE floating pt compare unordered | **SIGFPE** | **FPE_FLTBSUN_TRAP** |
| IEEE floating pt signaling NaN | **SIGFPE** | **FPE_FLTNAN_TRAP** |

**ADDR**

The *addr* signal handler parameter is defined as follows:

| Signal | Code | Addr |
|---|---|---|
| Sun: | | |
| **SIGILL** | Any | address of faulted instruction |
| **SIGEMT** | Any | address of faulted instruction |
| **SIGFPE** | Any | address of faulted instruction |
| **SIGBUS** | **BUS_HWERR** | address that caused fault |
| **SIGSEGV** | Any | address that caused fault |
| SPARC: | | |
| **SIGBUS** | **BUS_ALIGN** | address of faulted instruction |
| MC680X0: | | |
| **SIGBUS** | **BUS_ALIGN** | address that caused fault |

The accuracy of *addr* is machine dependent. For example, certain machines may supply an address that is on the same page as the address that caused the fault. If an appropriate *addr* cannot be computed it will be set to **SIG_NOADDR**.

**RETURN VALUE**

A 0 value indicated that the call succeeded. A −1 return value indicates an error occurred and **errno** is set to indicate the reason.

**ERRORS**

**sigvec( )** will fail and no new signal handler will be installed if one of the following occurs:

| | |
|---|---|
| EFAULT | Either *vec* or *ovec* is not a NULL pointer and points to memory that is not a valid part of the process address space. |
| EINVAL | *Sig* is not a valid signal number. |
| EINVAL | An attempt was made to ignore or supply a handler for **SIGKILL** or **SIGSTOP**. |
| EINVAL | An attempt is made to ignore **SIGCONT** (by default **SIGCONT** is ignored). |

**SEE ALSO**

**execve**(2), **fcntl**(2V), **fork**(2), **getrlimit**(2), **getitimer**(2), **ioctl**(2), **kill**(2V), **ptrace**(2), **read**(2V), **sigblock**(2), **sigpause**(2), **sigsetmask**(2), **sigstack**(2), **setjmp**(3), **signal**(3), **umask**(2), **vfork**(2), **wait**(2), **write**(2V), **streamio**(4), **termio**(4), **win**(4S), **lockd**(8C)

NAME

socket – create an endpoint for communication

SYNOPSIS

**#include <sys/types.h>**
**#include <sys/socket.h>**

**int socket(domain, type, protocol)**
**int domain, type, protocol;**

DESCRIPTION

**socket()** creates an endpoint for communication and returns a descriptor.

The *domain* parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file **<sys/socket.h>**. The currently understood formats are

| | |
|---|---|
| **PF_UNIX** | (UNIX system internal protocols), |
| **PF_INET** | (ARPA Internet protocols), and |
| **PF_IMPLINK** | (IMP "host at IMP" link layer). |

The socket has the indicated *type*, which specifies the semantics of communication. Currently defined types are:

**SOCK_STREAM**
**SOCK_DGRAM**
**SOCK_RAW**
**SOCK_SEQPACKET**
**SOCK_RDM**

A **SOCK_STREAM** type provides sequenced, reliable, two-way connection based byte streams. An out-of-band data transmission mechanism may be supported. A **SOCK_DGRAM** socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A **SOCK_SEQPACKET** socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is protocol specific, and presently not implemented for any protocol family. **SOCK_RAW** sockets provide access to internal network interfaces. The types **SOCK_RAW**, which is available only to the super-user, and **SOCK_RDM**, for which no implementation currently exists, are not described here.

The *protocol* specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place; see **protocols**(5).

Sockets of type **SOCK_STREAM** are full-duplex byte streams, similar to pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a **connect**(2) call. Once connected, data may be transferred using **read**(2V) and **write**(2V) calls or some variant of the **send**(2) and **recv**(2) calls. When a session has been completed a **close**(2) may be performed. Out-of-band data may also be transmitted as described in **send**(2) and received as described in **recv**(2).

The communications protocols used to implement a **SOCK_STREAM** insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with −1 returns and with ETIMEDOUT as the specific code in the global variable **errno**. The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for a

extended period (for instance 5 minutes). A **SIGPIPE** signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

**SOCK_SEQPACKET** sockets employ the same system calls as **SOCK_STREAM** sockets. The only difference is that **read**(2V) calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

**SOCK_DGRAM** and **SOCK_RAW** sockets allow sending of datagrams to correspondents named in **send**(2) calls. Datagrams are generally received with **recv**(2), which returns the next datagram with its return address.

An **fcntl**(2V) call can be used to specify a process group to receive a **SIGURG** signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notification of I/O events with **SIGIO** signals.

The operation of sockets is controlled by socket level *options*. These options are defined in the file **<sys/socket.h>**. **getsockopt**(2) and **getsockopt**(2) are used to set and get options, respectively.

## RETURN VALUE

A −1 is returned if an error occurs, otherwise the return value is a descriptor referencing the socket.

## ERRORS

The **socket()** call fails if:

| | |
|---|---|
| EPROTONOSUPPORT | The protocol type or the specified protocol is not supported within this domain. |
| EMFILE | The per-process descriptor table is full. |
| ENFILE | The system file table is full. |
| EACCESS | Permission to create a socket of the specified type and/or protocol is denied. |
| ENOBUFS | Insufficient buffer space is available. The socket cannot be created until sufficient resources are freed. |
| EPROTOTYPE | The protocol is the wrong type for the socket. |

## SEE ALSO

**accept**(2), **bind**(2), **close**(2), **connect**(2), **fcntl**(2V), **getsockname**(2), **getsockopt**(2), **ioctl**(2), **listen**(2), **read**(2V), **recv**(2), **select**(2), **send**(2), **shutdown**(2), **socketpair**(2), **write**(2V), **protocols**(5)

*Inter-Process Communication Primer* in *Network Programming*

## NAME

socketpair – create a pair of connected sockets

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

socketpair(d, type, protocol, sv)
int d, type, protocol;
int sv[2];
```

## DESCRIPTION

The **socketpair**( ) system call creates an unnamed pair of connected sockets in the specified address family *d*, of the specified *type* and using the optionally specified *protocol*. The descriptors used in referencing the new sockets are returned in *sv*[0] and *sv*[1]. The two sockets are indistinguishable.

## DIAGNOSTICS

**socketpair**( ) returns a −1 on failure, otherwise it returns the number of the second file descriptor it creates.

## ERRORS

The call succeeds unless:

| | |
|---|---|
| EMFILE | Too many descriptors are in use by this process. |
| EAFNOSUPPORT | The specified address family is not supported on this machine. |
| EPROTONOSUPPORT | The specified protocol is not supported on this machine. |
| EOPNOSUPPORT | The specified protocol does not support creation of socket pairs. |
| EFAULT | The address *sv* does not specify a valid part of the process address space. |

## SEE ALSO

**read**(2V), **write**(2V), **pipe**(2)

## BUGS

This call is currently implemented only for the **AF_UNIX** address family.

## NAME

stat, lstat, fstat – get file status

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(path, buf)
char *path;
struct stat *buf;

int lstat(path, buf)
char *path;
struct stat *buf;

int fstat(fd, buf)
int fd;
struct stat *buf;
```

## DESCRIPTION

**stat()** obtains information about the file named by *path*. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

**lstat()** is like **stat()** except in the case where the named file is a symbolic link, in which case **lstat()** returns information about the link, while **stat()** returns information about the file the link references.

*fstat* obtains the same information about an open file referenced by the argument descriptor, such as would be obtained by an **open**(2V) call.

*buf* is a pointer to a **stat()** structure into which information is placed concerning the file. A **stat()** structure includes the following members:

```
dev_t     st_dev;     /* device inode resides on */
ino_t     st_ino;     /* this inode's number */
u_short   st_mode;    /* protection */
short     st_nlink;   /* number of hard links to the file */
short     st_uid;     /* user ID of owner */
short     st_gid;     /* group ID of owner */
dev_t     st_rdev;    /* the device type, for inode that is device */
off_t     st_size;    /* total size of file, in bytes */
time_t    st_atime;   /* file last access time */
time_t    st_mtime;   /* file last modify time */
time_t    st_ctime;   /* file last status change time */
long      st_blksize; /* optimal blocksize for file system i/o ops */
long      st_blocks;  /* actual number of blocks allocated */
```

**st_atime** Time when file data was last read or modified. Changed by the following system calls: **mknod**(2), **utimes**(2), **read**(2V), **write**(2V), and **truncate**(2). For reasons of efficiency, **st_atime** is not set when a directory is searched, although this would be more logical.

**st_mtime** Time when data was last modified. It is not set by changes of owner, group, link count, or mode. Changed by the following system calls: **mknod**(2), **utimes**(2), **write**(2V).

**st_ctime** Time when file status was last changed. It is set both both by writing and changing the inode. Changed by the following system calls: **chmod**(2) **chown**(2), **link**(2), **mknod**(2), **rename**(2), **unlink**(2), **utimes**(2), **write**(2V), **truncate**(2).

The status information word **st_mode** has bits:

```
#define  S_IFMT     0170000     /* type of file */
#define  S_IFIFO    0010000     /* fifo special */
#define  S_IFCHR    0020000     /* character special */
```

```
#define  S_IFDIR     0040000    /* directory */
#define  S_IFBLK     0060000    /* block special */
#define  S_IFREG     0100000    /* regular file */
#define  S_IFLNK     0120000    /* symbolic link */
#define  S_IFSOCK    0140000    /* socket */
#define  S_ISUID     0004000    /* set user id on execution */
#define  S_ISGID     0002000    /* set group id on execution */
#define  S_ISVTX     0001000    /* save swapped text even after use */
#define  S_IREAD     0000400    /* read permission, owner */
#define  S_IWRITE    0000200    /* write permission, owner */
#define  S_IEXEC     0000100    /* execute/search permission, owner */
```

The mode bits 0000070 and 0000007 encode group and others permissions (see **chmod**(2)).

## RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

## ERRORS

**stat**( ) and **lstat**( ) will fail if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The file referred to by *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EFAULT | *buf* or *path* points to an invalid address. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

**fstat**( ) will fail if one or both of the following are true:

| | |
|---|---|
| EBADF | *fd* is not a valid open file descriptor. |
| EFAULT | *buf* points to an invalid address. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

## CAVEAT

The **st_atime** and **st_mtime** fields of the **stat**( ) are *not* contiguous. Programs that depend on them being contiguous (in calls to **utimes**(2) or **utime**(3C)) will not work.

## SEE ALSO

**chmod**(2), **chown**(2), **link**(2), **mknod**(2), **read**(2V), **readlink**(2), **rename**(2), **truncate**(2), **unlink**(2), **utimes**(2), **write**(2V)

**NAME**

statfs – get file system statistics

**SYNOPSIS**

```
#include <sys/vfs.h>

int statfs(path, buf)
char *path;
struct statfs *buf;

int fstatfs(fd, buf)
int fd;
struct statfs *buf;
```

**DESCRIPTION**

**statfs( )** returns information about a mounted file system. *path* is the path name of any file within the mounted filesystem. *buf* is a pointer to a **statfs( )** structure defined as follows:

```
typedef struct {
        long    val[2];
} fsid_t;

struct statfs {
        long    f_type;     /* type of info, zero for now */
        long    f_bsize;    /* fundamental file system block size */
        long    f_blocks;   /* total blocks in file system */
        long    f_bfree;    /* free blocks */
        long    f_bavail;   /* free blocks available to non-super-user */
        long    f_files;    /* total file nodes in file system */
        long    f_ffree;    /* free file nodes in fs */
        fsid_t  f_fsid;     /* file system id */
        long    f_spare[7]; /* spare for later */
};
```

Fields that are undefined for a particular file system are set to –1. *fstatfs* returns the same information about an open file referenced by descriptor *fd*.

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, –1 is returned and the global variable **errno** is set to indicate the error.

**ERRORS**

**statfs( )** fails if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The file referred to by *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EFAULT | *buf* or *path* points to an invalid address. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

*fstatfs* fails if one or both of the following are true:

| | |
|---|---|
| EBADF | *fd* is not a valid open file descriptor. |

| | |
|---|---|
| EFAULT | *buf* points to an invalid address. |
| EIO | An I/O error occurred while reading from the file system. |

**NAME**

swapon – add a swap device for interleaved paging/swapping

**SYNOPSIS**

**int swapon(special)**
**char *special;**

**DESCRIPTION**

**swapon( )** makes the block device *special* available to the system for allocation for paging and swapping. The names of potentially available devices are known to the system and defined at system configuration time. The size of the swap area on *special* is calculated at the time the device is first made available for swapping.

**RETURN VALUE**

If an error has occurred, a value of –1 is returned and **errno** is set to indicate the error.

**ERRORS**

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *special* is not a directory. |
| ENAMETOOLONG | The length of a component of *special* exceeds 255 characters, or the length of *special* exceeds 1023 characters. |
| ENOENT | The device referred to by *special* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *special*. |
| ELOOP | Too many symbolic links were encountered in translating *special*. |
| EPERM | The caller is not the super-user. |
| ENOTBLK | The file referred to by *special* is not a block device. |
| EBUSY | The device referred to by *special* has already been made available for swapping. |
| ENODEV | The device referred to by *special* was not configured into the system as a swap device. |
| ENXIO | The major device number of the device referred to by *special* is out of range (this indicates no device driver exists for the associated hardware). |
| EIO | An I/O error occurred while reading from or writing to the file system or opening the swap device. |
| EFAULT | *special* points outside the process's address space. |

**SEE ALSO**

**config**(8), **swapon**(8)

**BUGS**

There is no way to stop swapping on a disk so that the pack may be dismounted.

This call will be upgraded in future versions of the system.

## NAME

symlink – make symbolic link to a file

## SYNOPSIS

**int symlink(name1, name2)**
**char *name1, *name2;**

## DESCRIPTION

A symbolic link *name2* is created to *name1* (*name2* is the name of the file created, *name1* is the string used in creating the symbolic link). Either name may be an arbitrary path name; the files need not be on the same file system.

The file that the symbolic link points to is used when an **open**(2V) operation is performed on the link. A **stat**(2) on a symbolic link returns the linked-to file, while an **lstat** returns information about the link itself. This can lead to surprising results when a symbolic link is made to a directory. To avoid confusion in programs, the **readlink**(2) call can be used to read the contents of a symbolic link.

## RETURN VALUE

Upon successful completion, a zero value is returned. If an error occurs, the error code is stored in **errno** and a −1 value is returned.

## ERRORS

The symbolic link is made unless one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *name2* is not a directory. |
| ENAMETOOLONG | The length of a component of either *name1* or *name2* exceeds 255 characters, or the length of either *name1* or *name2* exceeds 1023 characters. |
| ENOENT | A component of the path prefix of *name2* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *name2*. |
| ELOOP | Too many symbolic links were encountered in translating *name2*. |
| EEXIST | The file referred to by *name2* already exists. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EROFS | The file *name2* would reside on a read-only file system. |
| ENOSPC | The directory in which the entry for the new symbolic link is being placed cannot be extended because there is no space left on the file system containing the directory. |
| ENOSPC | The new symbolic link cannot be created because there is no space left on the file system which will contain the link. |
| ENOSPC | There are no free inodes on the file system on which the file is being created. |
| EDQUOT | The directory in which the entry for the new symbolic link is being placed cannot be extended because the user's quota of disk blocks on the file system containing the directory has been exhausted. |
| EDQUOT | The new symbolic link cannot be created because the user's quota of disk blocks on the file system which will contain the link has been exhausted. |
| EDQUOT | The user's quota of inodes on the file system on which the file is being created has been exhausted. |
| EFAULT | *name1* or *name2* points outside the process's allocated address space. |

## SEE ALSO

**ln**(1), **link**(2), **readlink**(2), **unlink**(2)

**NAME**

sync – update super-block

**SYNOPSIS**

**sync()**

**DESCRIPTION**

**sync()** writes out all information in core memory that should be on disk. This includes modified super blocks, modified inodes, and delayed block I/O.

**sync()** should be used by programs that examine a file system, for example **fsck**(8), **df**(1), etc. **sync()** is mandatory before a boot.

**SEE ALSO**

**fsync**(2), **cron**(8)

**BUGS**

The writing, although scheduled, is not necessarily complete upon return from **sync()**.

**NAME**

syscall – indirect system call

**SYNOPSIS**

**#include <sys/syscall.h>**

**int syscall(number, arg, ...)**

**DESCRIPTION**

**syscall()** performs the system call whose assembly language interface has the specified *number*, and arguments *arg* .... Symbolic constants for system calls can be found in the header file **<sys/syscall.h>**.

On Sun-2, Sun-3, and Sun-4 systems, the value of register **d0** after the system call is returned. On Sun386i systems, the value of register **%eax** is returned.

**SEE ALSO**

**intro**(2), **pipe**(2)

**DIAGNOSTICS**

When the C-bit is set, **syscall()** returns –1 and sets the external variable **errno** (see **intro**(2)).

**BUGS**

There is no way to simulate system calls such as **pipe**(2), which return values in register **d1** on Sun-2, Sun-3, and Sun-4 systems or in register **%edx** on Sun386i systems.

## NAME

truncate, ftruncate – set a file to a specified length

## SYNOPSIS

```
#include <sys/types.h>

int truncate(path, length)
char *path;
off_t length;

int ftruncate(fd, length)
int fd;
off_t length;
```

## DESCRIPTION

**truncate( )** causes the file referred to by *path* (or for **ftruncate( )** the object referred to by *fd*) to have a size equal to *length* bytes. If the file was previously longer than *length*, the extra bytes are removed from the file. If it was shorter, bytes between the old and new lengths are read as zeroes. With **ftruncate**, the file must be open for writing.

## RETURN VALUES

A value of 0 is returned if the call succeeds. If the call fails a −1 is returned, and the global variable **errno** specifies the error.

## ERRORS

**truncate( )** succeeds unless:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The file referred to by *path* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| EACCES | Write permission is denied for the file referred to by *path*. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EISDIR | The file referred to by *path* is a directory. |
| EROFS | The file referred to by *path* resides on a read-only file system. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EFAULT | *path* points outside the process's allocated address space. |

**ftruncate( )** succeeds unless:

| | |
|---|---|
| EINVAL | *fd* is not a valid descriptor of a file open for writing. |
| EINVAL | *fd* refers to a socket, not to a file. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

## SEE ALSO

**open**(2V)

## BUGS

These calls should be generalized to allow ranges of bytes in a file to be discarded.

**NAME**

umask – set file creation mode mask

**SYNOPSIS**

**int umask(numask)**

**int numask;**

**DESCRIPTION**

**umask**( ) sets the process's file mode creation mask to *numask* and returns the previous value of the mask. The low-order 9 bits of *numask* are used whenever a file is created, clearing corresponding bits in the file mode (see **chmod**(2)). This clearing allows each user to restrict the default access to his files.

The mask is inherited by child processes.

**RETURN VALUE**

The previous value of the file mode mask is returned by the call.

**SEE ALSO**

**chmod**(2), **mknod**(2), **open**(2V)

## NAME

uname – get name of current system

## SYNOPSIS

**#include <sys/utsname.h>**

**int uname(name)**

**struct utsname *name;**

## DESCRIPTION

Note: This system call is only available for use with the System V compatibility libraries. These are located in the directory **/usr/5lib**, and are compiled using the System V version of the C compiler, **/usr/5bin/cc**.

**uname()** stores information identifying the current system in the structure pointed to by *name*.

**uname()** uses the structure defined in **<sys/utsname.h>** whose members are:

```
char    sysname[9];
char    nodename[65];
char    release[9];
char    version[9];
char    machine[9];
```

**uname()** returns a null-terminated string giving the standard host name for the current processor in **nodename**. This name will be same as the name returned by the **gethostname**(2) system call. It also returns a null-terminated character string naming the current operating system in **sysname**. **release** and **version** further identify the operating system. **machine** contains a name that identifies the hardware of the current processor.

## FILES

**/usr/5lib**
**/usr/5bin/cc**

## SEE ALSO

**gethostname**(2), **uname**(1V)

**NAME**

unlink – remove directory entry

**SYNOPSIS**

**int unlink(path)**
**char *path;**

**DESCRIPTION**

**unlink()** removes the directory entry named by the path name pointed to by *path*. If this entry was the last link to the file, and no process has the file open, then all resources associated with the file are reclaimed. If, however, the file was open in any process, the actual resource reclamation is delayed until it is closed, even though the directory entry has disappeared.

**RETURN VALUE**

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

**ERRORS**

The **unlink()** succeeds unless:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *path* is not a directory. |
| ENAMETOOLONG | The length of a component of *path* exceeds 255 characters, or the length of *path* exceeds 1023 characters. |
| ENOENT | The file referred to by *path* does not exist. |
| EINVAL | The file referred to by *path* is the current directory, '.'. |
| EACCES | Search permission is denied for a component of the path prefix of *path*. |
| EACCES | Write permission is denied for the directory containing the link to be removed. |
| ELOOP | Too many symbolic links were encountered in translating *path*. |
| EPERM | The file referred to by *path* is a directory and the effective user ID of the process is not the super-user. |
| EBUSY | The entry to be unlinked is the mount point for a mounted file system. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EROFS | The file referred to by *path* resides on a read-only file system. |
| EFAULT | *path* points outside the process's allocated address space. |

**SEE ALSO**

**close**(2), **link**(2), **rmdir**(2)

**NAME**

unmount – remove a file system

**SYNOPSIS**

**unmount(name)**
**char *name;**

**DESCRIPTION**

**unmount()** announces to the system that the directory *name* is no longer to refer to the root of a mounted file system. The directory *name* reverts to its ordinary interpretation.

**RETURN VALUE**

**unmount()** returns 0 if the action occurred; −1 if if the directory is inaccessible or does not have a mounted file system, or if there are active files in the mounted file system.

**ERRORS**

**unmount()** may fail with one of the following errors:

| | |
|---|---|
| EPERM | The caller is not the super-user. |
| ENOTDIR | A component of the path prefix of *name* is not a directory. |
| EINVAL | *name* is not the root of a mounted file system. |
| EBUSY | A process is holding a reference to a file located on the file system. |
| ENAMETOOLONG | The length of a component of the path name exceeds 255 characters, or the length of the entire path name exceeds 1023 characters. |
| ENOENT | *name* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix. |
| EFAULT | *name* points outside the process's allocated address space. |
| ELOOP | Too many symbolic links were encountered in translating the path name. |
| EIO | An I/O error occurred while reading from or writing to the file system. |

**SEE ALSO**

**mount**(2), **mount**(8),

**BUGS**

The error codes are in a state of disarray; too many errors appear to the caller as one value.

NAME

utimes – set file times

SYNOPSIS

**#include <sys/types.h>**

**int utimes(file, tvp)**
**char *file;**
**struct timeval *tvp;**

DESCRIPTION

**utimes( )** sets the access and modification times of the file named by *file*.

If *tvp* is NULL, the access and modification times are set to the current time. A process must be the owner of the file or have write permission for the file to use **utimes( )** in this manner.

If *tvp* is not NULL, it is assumed to point to an array of two **timeval** structures. The access time is set to the value of the first member, and the modification time is set to the value of the second member. Only the owner of the file or the super-user may use **utimes( )** in this manner.

In either case, the *inode-changed* time of the file is set to the current time.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

ERRORS

**utimes( )** will fail if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *file* is not a directory. |
| ENAMETOOLONG | The length of a component of *file* exceeds 255 characters, or the length of *file* exceeds 1023 characters. |
| ENOENT | The file referred to by *file* does not exist. |
| EACCES | Search permission is denied for a component of the path prefix of *file*. |
| ELOOP | Too many symbolic links were encountered in translating *file*. |
| EPERM | The effective user ID of the process is not super-user and not the owner of the file, and *tvp* is not NULL. |
| EACCES | The effective user ID of the process is not super-user and not the owner of the file, write permission is denied for the file, and *tvp* is NULL. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EROFS | The file system containing the file is mounted read-only. |
| EFAULT | *file* or *tvp* points outside the process's allocated address space. |

SEE ALSO

**stat**(2)

NAME

vadvise – give advice to paging system

SYNOPSIS

**#include <sys/vadvise.h>**

**vadvise(param)**
**int param;**

DESCRIPTION

**vadvise()** is used to inform the system that process paging behavior merits special consideration. Parameters to **vadvise()** are defined in the file **<vadvise.h>**. Currently, two calls t **vadvise()** are implemented.

The call

**vadvise(VA_ANOM);**

advises that the paging behavior is not likely to be well handled by the system's default algorithm, since reference information is collected over macroscopic intervals (for instance, 10-20 seconds) will not serve to indicate future page references. The system in this case will choose to replace pages with little emphasis placed on recent usage, and more emphasis on referenceless circular behavior. It is *essential* that processes which have very random paging behavior (such as LISP during garbage collection of very large address spaces) call **vadvise**, as otherwise the system has great difficulty dealing with their page-consumptive demands.

The call

**vadvise(VA_NORM);**

restores default paging replacement behavior after a call to

**vadvise(VA_ANOM);**

BUGS

Will go away soon, being replaced by a per-page **vadvise()** facility.

## NAME

vfork – spawn new process in a virtual memory efficient way

## SYNOPSIS

**#include <vfork.h>**

**pid = vfork()**

**int pid;**

## DESCRIPTION

**vfork**( ) can be used to create new processes without fully copying the address space of the old process, which is horrendously inefficient in a paged environment. It is useful when the purpose of **fork**(2) would have been to create a new system context for an **execve**(2). **vfork**( ) differs from **fork**( ) in that the child borrows the parent's memory and thread of control until a call to **execve**(2) or an exit (either by a call to **exit**(2) or abnormally.) The parent process is suspended while the child is using its resources.

**vfork**( ) returns 0 in the child's context and (later) the *pid* of the child in the parent's context.

**vfork**( ) can normally be used just like **fork**. It does not work, however, to return while running in the child's context from the procedure which called **vfork**( ) since the eventual return from **vfork**( ) would then return to a no longer existent stack frame. Be careful, also, to call **_exit**( ) rather than **exit**( ) if you cannot *execve*, since **exit**( ) will flush and close standard I/O channels, and thereby mess up the parent processes standard I/O data structures. (Even with **fork**( ) it is wrong to call **exit**( ) since buffered data would then be flushed twice.)

On Sun-4 machines, the parent inherits the values of local and incoming argument registers from the child. Since this violates the usual data flow properties of procedure calls, the file **<vfork.h>** must be included in programs that are compiled using global optimization.

## SEE ALSO

**execve**(2), **exit**(2), **fork**(2), **ioctl**(2), **sigvec**(2), **wait**(2),

## DIAGNOSTICS

Same as for **fork**(2).

## BUGS

This system call will be eliminated when proper system sharing mechanisms are implemented. Users should not depend on the memory sharing semantics of **vfork**( ) as it will, in that case, be made synonymous to **fork**(2).

To avoid a possible deadlock situation, processes that are children in the middle of a **vfork**( ) are never sent **SIGTTOU** or **SIGTTIN** signals; rather, output or *ioctl*s are allowed and input attempts result in an EOF indication.

NAME

vhangup – virtually "hangup" the current control terminal

SYNOPSIS

**vhangup()**

DESCRIPTION

**vhangup()** is used by the initialization process **init**(8) (among others) to arrange that users are given "clean" terminals at login, by revoking access of the previous users' processes to the terminal. To affect this, **vhangup()** searches the system tables for references to the control terminal of the invoking process, revoking access permissions on each instance of the terminal that it finds. Further attempts to access the terminal by the affected processes will yield I/O errors (EBADF). Finally, a **SIGHUP** (hangup signal) is sent to the process group of the control terminal.

SEE ALSO

**init**(8)

BUGS

Access to the control terminal using **/dev/tty** is still possible.

This call should be replaced by an automatic mechanism that takes place on process exit.

NAME

wait, wait3, wait4, WIFSTOPPED, WIFSIGNALED, WIFEXITED – wait for process to terminate or stop

SYNOPSIS

```
#include <sys/wait.h>

int wait(statusp)
union wait *statusp;

int wait((union wait *)0)

#include <sys/time.h>
#include <sys/resource.h>

int wait3(statusp, options, rusage)
union wait *statusp;
int options;
struct rusage *rusage;

int wait4(pid, statusp, options, rusage)
int pid;
union wait *statusp;
int options;
struct rusage *rusage;

WIFSTOPPED(status)
union wait status;

WIFSIGNALED(status)
union wait status;

WIFEXITED(status)
union wait status;
```

DESCRIPTION

**wait()** delays its caller until a signal is received or one of its child processes terminates or stops due to tracing. If any child has died or stopped due to tracing and this has not been reported using **wait**, return is immediate, returning the process ID and exit status of one of those children. If that child had died, it is discarded. If there are no children, return is immediate with the value –1 returned. If there are only running or stopped but reported children, the calling process is blocked.

If *status* is not a NULL pointer, then on return from a successful **wait()** call the status of the child process whose process ID is the return value of **wait()** is stored in the **wait()** union pointed to by *status*. The **w_status** member of that union is an **int**; it indicates the cause of termination and other information about the terminated process in the following manner:

- If the low-order 8 bits of **w_status** are equal to 0177, the child process has stopped; the 8 bits higher up from the low-order 8 bits of **w_status** contain the number of the signal that caused the process to stop. See **ptrace**(2) and **sigvec**(2).
- If the low-order 8 bits of **w_status** are non-zero and are not equal to 0177, the child process terminated due to a signal; the low-order 7 bits of **w_status** contain the number of the signal that terminated the process. In addition, if the low-order seventh bit of **w_status** (that is, bit 0200) is set, a "core image" of the process was produced; see **sigvec**(2).
- Otherwise, the child process terminated due to an **exit()** call; the 8 bits higher up from the low-order 8 bits of **w_status** contain the low-order 8 bits of the argument that the child process passed to **exit**; see **exit**(2).

Other members of the **wait()** union can be used to extract this information more conveniently:

- If the **w_stopval** member has the value **WSTOPPED**, the child process has stopped; the value of the **w_stopsig** member is the signal that stopped the process.
- If the **w_termsig** member is non-zero, the child process terminated due to a signal; the value of the **w_termsig** member is the number of the signal that terminated the process. If the **w_coredump** member is non-zero, a core dump was produced.
- Otherwise, the child process terminated due to an **exit()** call; the value of the **w_retcode** member is the low-order 8 bits of the argument that the child process passed to **exit**.

The other members of the **wait()** union merely provide an alternate way of analyzing the status. The value stored in the **w_status** field is compatible with the values stored by other versions of the UNIX system, and an argument of type **int *** may be provided instead of an argument of type **union wait *** for compatibility with those versions.

**wait3()** is an alternate interface that allows both non-blocking status collection and the collection of the status of children stopped by any means. The *status* parameter is defined as above. The *options* parameter is used to indicate the call should not block if there are no processes that have status to report (**WNOHANG**), and/or that children of the current process that are stopped due to a **SIGTTIN**, **SIGTTOU**, **SIGTSTP**, or **SIGSTOP** signal are eligible to have their status reported as well (**WUNTRACED**). A terminated child is discarded after it reports status, and a stopped process will not report its status more than once. If *rusage* is not a NULL pointer, a summary of the resources used by the terminated process and all its children is returned. (This information is currently not available for stopped processes.)

When the WNOHANG option is specified and no processes have status to report, **wait3()** returns 0. The **WNOHANG** and **WUNTRACED** options may be combined by ORing the two values.

**wait4()** is another alternate interface. With a *pid* argument of 0, it is equivalent to **wait3**. If *pid* has a nonzero value, then **wait4()** returns status only for the indicated process ID, but not for any other child processes.

**WIFSTOPPED**, **WIFSIGNALED**, **WIFEXITED**, are macros that take an argument *status*, of type '**union wait**', as returned by **wait**, **wait2**, **wait3**, or **wait4**. **WIFSTOPPED** evaluates to true (1) when the process for which the **wait()** call was made is stopped, or to false (0) otherwise. **WIFSIGNALED** evaluates to true when the process was terminated with a signal. **WIFEXITED** evaluates to true whe the process exited by using an **exit**(2) call.

## NOTES

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

**wait**, **wait3**, and **wait4()** are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the **SV_INTERRUPT** bit is set in the flags for that signal.

## RETURN VALUE

If **wait()** returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

**wait3()** and **wait4()** return 0 if **WNOHANG** is specified and there are no stopped or exited children, and return the process ID of the child process if they return due to a stopped or terminated child process. Otherwise, they return a value of −1 and set **errno** to indicate the error.

## ERRORS

**wait**, **wait3**, or **wait4** will fail and return immediately if one or more of the following are true:

| | |
|---|---|
| ECHILD | The calling process has no existing unwaited-for child processes. |
| EFAULT | The *status* or *rusage* arguments point to an illegal address. |

**wait, wait3**, and **wait4** will terminate prematurely, return −1, and set **errno** t EINTR upon the arrival of a signal whose **SV_INTERRUPT** bit in its flags field is set (see **sigvec**(2) and **siginterrupt**(3)). **signal**(3V), in the System V compatibility library, sets this bit for any signal it catches.

**SEE ALSO**

**exit**(2), **getrusage**(2), **ptrace**(2), **sigvec**(2), **siginterrupt**(3), **signal**(3)

**WARNINGS**

Calls to **wait** with an argument of **0** should be cast to type '**union**wait*', as in:

**wait( (union wait *) 0)**

Otherwise **lint**(1V) will complain.

**NAME**

write, writev – write output

**SYNOPSIS**

```
int write(d, buf, nbytes)
int d;
char *buf;
int nbytes;

#include <sys/types.h>
#include <sys/uio.h>

int writev(d, iov, iovcnt)
int d;
struct iovec *iov;
int iovcnt;
```

**DESCRIPTION**

**write()** attempts to write *nbytes* of data to the object referenced by the descriptor *d* from the buffer pointed to by *buf*. **writev()** performs the same action, but gathers the output data from the *iovcnt* buffers specified by the members of the *iov* array: iov[0], iov[1], ..., iov[iovcnt – 1].

For **writev**, the *iovec* structure is defined as

```
struct iovec {
        caddr_t iov_base;
        int     iov_len;
};
```

Each *iovec* entry specifies the base address and length of an area in memory from which data should be written. **writev()** will always write a complete area before proceeding to the next.

On objects capable of seeking, the **write()** starts at a position given by the pointer associated with *d*, see **lseek**(2). Upon return from **write**, the pointer is incremented by the number of bytes actually written.

Objects that are not capable of seeking always write from the current position. The value of the pointer associated with such an object is undefined.

If the **O_APPEND** flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

For regular files, if the **O_SYNC** flag of the file status flags is set, the write will not return until both the file data and file status have been physically updated. This function is for special applications that require extra reliability at the cost of performance. For block special files, if **O_SYNC** is set, the write will not return until the data has been physically updated.

If the real user is not the super-user, then **write()** clears the set-user-id bit on a file. This prevents penetration of system security by a user who "captures" a writable set-user-id file owned by the super-user.

For STREAMS (see **intro**(2)) files, the operation of **write()** and **writev()** are determined by the values of the minimum and maximum packet sizes accepted by the *stream*. These values are contained in the topmost *stream* module. Unless the user pushes (see **I_PUSH** in **streamio**(4)) the topmost module, these values can not be set or tested from user level. If the total number of bytes to be written falls within the packet size range, that many bytes will be written. If the total number of bytes to be written does not fall within the range and the minimum packet size value is zero, **write()** and **writev()** will break the data to be written into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If the total number of bytes to be written does not fall within the range and the minimum value is non-zero, **write()** and **writev()** will fail with **errno** set to ERANGE. Writing a zero-length buffer (the total number of bytes to be written is zero) sends zero bytes with zero returned.

When a descriptor or the object it refers to is marked for non-blocking I/O, and the descriptor refers to an object subject to flow control, such as a socket, a pipe (or FIFO), or a *stream*, **write()** and **writev()** may write fewer bytes than requested; the return value must be noted, and the remainder of the operation should be retried when possible. If such an object's buffers are full, so that it cannot accept any data, then:

- If the object the descriptor is associated with is marked for 4.2BSD-style non-blocking I/O (with the **FIONBIO** **ioctl**(2), or an **fcntl()** using the **FNDELAY** flag from **<sys/file.h>** or the **O_NDELAY** flag from **<sys/fcntl.h>** in the 4.2BSD environment), the write will return −1 and **errno** will be set to EWOULDBLOCK.
- If the descriptor is marked for System V-style non-blocking I/O (with an **fcntl()** using the **FNBIO** flag from **<sys/file.h>** or the **O_NDELAY** flag from **<sys/fcntl.h>** in the System V environment), and does not refer to a *stream*, the write will return 0.
- If the descriptor is marked for System V-style non-blocking I/O, and refers to a *stream*, the write will return −1 and **errno** will be set to EAGAIN.
- If neither the descriptor nor the object it refers to are marked for non-blocking I/O, the write will block until space becomes available.

## RETURN VALUE

Upon successful completion the number of bytes actually writen is returned. Otherwise a −1 is returned and the global variable **errno** is set to indicate the error.

## ERRORS

**write()** and **writev()** will fail and the file pointer will remain unchanged if one or more of the following are true:

| | |
|---|---|
| EBADF | *d* is not a valid descriptor open for writing. |
| EPIPE | An attempt is made to write to a pipe that is not open for reading by any process (or to a socket of type **SOCK_STREAM** that is connected to a peer socket.) Note: an attempted write of this kind will also cause you to receive a **SIGPIPE** signal from the kernel. If you've not made a special provision to catch or ignore this signal, your process will die. |
| EFBIG | An attempt was made to write a file that exceeds the process's file size limit or the maximum file size. |
| EFAULT | Part of *iov* or data to be written to the file points outside the process's allocated address space. |

The call is forced to terminate prematurely due to the arrival of a signal whose **SV_INTERRUPT** bit in **sv_flags** is set (see **sigvec**(2)). **signal**(3V), in the System V compatibility library, sets this bit for any signal it catches.

| | |
|---|---|
| EINVAL | The *stream* is linked below a multiplexor. |
| EINVAL | The pointer associated with *d* was negative. |
| ENOSPC | There is no free space remaining on the file system containing the file. |
| EDQUOT | The user's quota of disk blocks on the file system containing the file has been exhausted. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ENXIO | A hangup occurred on the *stream* being written to. |
| ERANGE | *d* refers to a *stream*, the total number of bytes to be written is outside the minimum and maximum write range, and the minimum value is non-zero. |
| EWOULDBLOCK | The file was marked for 4.2BSD-style non-blocking I/O, and no data could be written immediately. |
| EAGAIN | The descriptor referred to a *stream*, was marked for System V-style non-blocking |

I/O, and no data could be written immediately.

In addition, **writev**( ) may return one of the following errors:

EINVAL *iovcnt* was less than or equal to 0, or greater than 16.

EINVAL One of the **iov_len** values in the *iov* array was negative.

EINVAL The sum of the **iov_len** values in the *iov* array overflowed a 32-bit integer.

A write to a STREAMS file can fail if an error message has been received at the stream head. In this case, **errno** is set to the value included in the error message.

**SEE ALSO**

**dup**(2), **fcntl**(2V), **intro**(2), **ioctl**(2), **lseek**(2), **open**(2V), **pipe**(2), **select**(2), **sigvec**(2), **signal**(3V)

NAME

Intro – introduction to user-level library functions

DESCRIPTION

Section 3 describes user-level library routines. In this release, most user-library routines are listed in alphabetical order regardless of their subsection headings. (This eliminates having to search through several subsections of the manual.) However, due to their special-purpose nature, the routines from the following libraries are broken out into the indicated subsections:

- The Lightweight Processes Library, in subsection 3L.
- The RPC Services Library, in subsection 3R.
- The System V Compatibility Library, in subsection 3V. This library contains System V versions of functions that are not yet merged into the standard Sun libraries. To use them functions, compile programs with **/usr/5bin/cc**, instead of **/usr/bin/cc**.

The main C library, **/usr/lib/libc.a**, contains many of the functions described in this section, along with entry points for the system calls described in Section 2. This library also includes the Internet networking routines listed under the 3N subsection heading, and routines provided for compatibility with other UNIX operating systems, listed under 3C. Functions associated with the "standard I/O library" are listed under 3S.

User-level routines for access to data structures within the kernel and other processes are listed under 3K. To use these functions, compile programs with the **–lkvm** option for the C compiler, **cc**(1V).

Math library functions are listed under 3M. To use them, compile programs with the the **–lm cc**(1V) option.

Various specialized libraries, the routines they contain, and the compiler options needed to link with them, are listed under 3X.

FILES

| | |
|---|---|
| **/usr/lib/libc.a** | C Library (2, 3, 3N and 3C) |
| **/usr/lib/lib*.a** | other "standard" C libraries |
| **/usr/lib/lib*.a** | special-purpose C libraries |
| **/usr/5bin/cc** | |

SEE ALSO

**cc**(1V), **ld**(1), **nm**(1), **intro**(2)

LIST OF LIBRARY FUNCTIONS

| Name | Appears on Page | Description |
|---|---|---|
| – | **bstring**(3) | bit and byte string operations |
| – | **byteorder**(3N) | convert values between host and network byte order |
| – | **ctime**(3) | convert date and time |
| – | **ctype**(3) | character classification and conversion macros and functions |
| – | **curses**(3X) | cursor addressing and screen display library |
| – | **dbm**(3X) | data base subroutines |
| – | **directory**(3) | directory operations |
| – | **ethers**(3N) | Ethernet address mapping operations |
| – | **inet**(3N) | Internet address manipulation |
| – | **intro**(3L) | introduction to the lightweight process library (LWP) |
| – | **intro**(3M) | introduction to mathematical library functions |
| – | **intro**(3R) | introduction to RPC service library and protocols |
| – | **intro**(3V) | introduction to System V functions |
| – | **mp**(3X) | multiple precision integer arithmetic |
| – | **ndbm**(3) | data base subroutines |
| – | **plot**(3X) | graphics interface |

| | | |
|---|---|---|
| – | **rpc**(3N) | library routines for remote procedure calls |
| – | **string**(3) | string operations |
| – | **termcap**(3X) | terminal independent operation routines |
| – | **values**(3) | machine-dependent values |
| – | **xdr**(3N) | library routines for external data representation |
| **_crypt()** | **crypt**(3) | password and data encryption |
| **a64l()** | **a64l**(3) | convert between long integer and base-64 ASCII string |
| **abort()** | **abort**(3) | generate a fault |
| **abs()** | **abs**(3) | integer absolute value |
| **addexportent()** | **exportent**(3) | get exported file system information |
| **addexportent()** | **exportent**(3) | get exported file system information |
| **addmntent()** | **getmntent**(3) | get file system descriptor file entry |
| **addmntent()** | **getmntent**(3) | get file system descriptor file entry |
| **alloca()** | **malloc**(3) | memory allocator |
| **alloca()** | **malloc**(3) | memory allocator |
| **alphasort()** | **scandir**(3) | scan a directory |
| **alphasort()** | **scandir**(3) | scan a directory |
| **arc()** | **plot**(3X) | graphics interface |
| **asctime()** | **ctime**(3) | convert date and time |
| **assert()** | **assert**(3) | program verification |
| **atof()** | **strtod**(3) | convert string to double-precision number |
| **atoi()** | **strtol**(3) | convert string to integer |
| **atol()** | **strtol**(3) | convert string to integer |
| **audit()** | **getacinfo**(3) | get audit control file information |
| **audit_args()** | **audit_args**(3) | produce text audit message |
| **audit_text()** | **audit_args**(3) | produce text audit message |
| **auth_destroy()** | **rpc**(3N) | RPC services routines |
| **authdes_create()** | **rpc**(3N) | RPC services routines |
| **authdes_getcred()** | **rpc**(3N) | RPC services routines |
| **authnon_create()** | **rpc**(3N) | RPC services routines |
| **authunix_create()** | **rpc**(3N) | RPC services routines |
| **authunix_create_default()** | **rpc**(3N) | RPC services routines |
| **bcmp()** | **bstring**(3) | bit and byte string operations |
| **bcopy()** | **bstring**(3) | bit and byte string operations |
| **bindresvport()** | **bindresvport**(3N) | bind a socket to a privileged IP port |
| **bsearch()** | **bsearch**(3) | binary search a sorted table |
| **bzero()** | **bstring**(3) | bit and byte string operations |
| **calloc()** | **malloc**(3) | memory allocator |
| **callrpc()** | **rpc**(3N) | RPC services routines |
| **cbc_crypt()** | **des_crypt**(3) | fast DES encryption |
| **cfree()** | **malloc**(3) | memory allocator |
| **circle()** | **plot**(3X) | graphics interface |
| **clearerr()** | **ferror**(3S) | stream status inquiries |
| **clnt_broadcast()** | **rpc**(3N) | RPC services routines |
| **clnt_call()** | **rpc**(3N) | RPC services routines |
| **clnt_destroy()** | **rpc**(3N) | RPC services routines |
| **clnt_freeres()** | **rpc**(3N) | RPC services routines |
| **clnt_geterr()** | **rpc**(3N) | RPC services routines |
| **clnt_pcreateerror()** | **rpc**(3N) | RPC services routines |
| **clnt_perrno()** | **rpc**(3N) | RPC services routines |
| **clnt_perror()** | **rpc**(3N) | RPC services routines |
| **clnt_sperrno()** | **rpc**(3N) | RPC services routines |
| **clnt_sperror()** | **rpc**(3N) | RPC services routines |

| | | |
|---|---|---|
| **clntraw_create()** | **rpc**(3N) | RPC services routines |
| **clnttcp_create()** | **rpc**(3N) | RPC services routines |
| **clntudp_create()** | **rpc**(3N) | RPC services routines |
| **clock()** | **clock**(3C) | report CPU time used |
| **closedir()** | **directory**(3) | directory operations |
| **closelog()** | **syslog**(3) | control system log |
| **closepl()** | **plot**(3X) | graphics interface |
| **cont()** | **plot**(3X) | graphics interface |
| **control()** | **getacinfo**(3) | get audit control file information |
| **crypt()** | **crypt**(3) | password and data encryption |
| **ctermid()** | **ctermid**(3S) | generate filename for terminal |
| **cuserid()** | **cuserid**(3S) | get character login name of the user |
| **dbm_clearerr()** | **ndbm**(3) | data base subroutines |
| **dbm_close()** | **ndbm**(3) | data base subroutines |
| **dbm_delete()** | **ndbm**(3) | data base subroutines |
| **dbm_error()** | **ndbm**(3) | data base subroutines |
| **dbm_fetch()** | **ndbm**(3) | data base subroutines |
| **dbm_firstkey()** | **ndbm**(3) | data base subroutines |
| **dbm_nextkey()** | **ndbm**(3) | data base subroutines |
| **dbm_open()** | **ndbm**(3) | data base subroutines |
| **dbm_store()** | **ndbm**(3) | data base subroutines |
| **dbminit()** | **dbm**(3X) | data base subroutines |
| **decimal_to_double()** | **decimal_to_floating**(3) | convert decimal record to floating-point value |
| **decimal_to_extended()** | **decimal_to_floating**(3) | convert decimal record to floating-point value |
| **decimal_to_single()** | **decimal_to_floating**(3) | convert decimal record to floating-point value |
| **delete()** | **dbm**(3X) | data base subroutines |
| **des_crypt()** | **des_crypt**(3) | fast DES encryption |
| **des_setparity()** | **des_crypt**(3) | fast DES encryption |
| **dn_comp()** | **resolver**(3) | resolver routines |
| **dn_expand()** | **resolver**(3) | resolver routines |
| **double_to_decimal()** | **floating_to_decimal**(3) | convert floating-point value to decimal record |
| **drand48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **dysize()** | **ctime**(3) | convert date and time |
| **ecb_crypt()** | **des_crypt**(3) | fast DES encryption |
| **econvert()** | **econvert**(3) | output conversion |
| **ecvt()** | **econvert**(3) | output conversion |
| **edata()** | **end**(3) | last locations in program |
| **encrypt()** | **crypt**(3) | password and data encryption |
| **end()** | **end**(3) | last locations in program |
| **endac()** | **getacinfo**(3) | get audit control file information |
| **endexportent()** | **exportent**(3) | get exported file system information |
| **endfsent()** | **getfsent**(3) | get file system descriptor file entry |
| **endgraent()** | **getgraent**(3) | get group adjunct file entry |
| **endgrent()** | **getgrent**(3) | get group file entry |
| **endhostent()** | **gethostent**(3N) | get network host entry |
| **endmntent()** | **getmntent**(3) | get file system descriptor file entry |
| **endnetent()** | **getnetent**(3N) | get network entry |
| **endnetgrent()** | **getnetgrent**(3N) | get network group entry |
| **endprotoent()** | **getprotoent**(3N) | get protocol entry |
| **endpwaent()** | **getpwaent**(3) | get password adjunct file entry |
| **endpwent()** | **getpwent**(3) | get password file entry |
| **endservent()** | **getservent**(3N) | get service entry |
| **endttyent()** | **getttyent**(3) | get ttytab file entry |

| | | |
|---|---|---|
| **endusershell()** | **getusershell(3)** | get legal user shells |
| **erand48()** | **drand48(3)** | generate uniformly distributed pseudo-random numbe |
| **erase()** | **plot(3X)** | graphics interface |
| **errno()** | **perror(3)** | system error messages |
| **etext()** | **end(3)** | last locations in program |
| **ether_aton()** | **ethers(3N)** | Ethernet address mapping operations |
| **ether_hostton()** | **ethers(3N)** | Ethernet address mapping operations |
| **ether_line()** | **ethers(3N)** | Ethernet address mapping operations |
| **ether_ntoa()** | **ethers(3N)** | Ethernet address mapping operations |
| **ether_ntohost()** | **ethers(3N)** | Ethernet address mapping operations |
| **execl()** | **execl(3)** | execute a file |
| **execle()** | **execl(3)** | execute a file |
| **execlp()** | **execl(3)** | execute a file |
| **execv()** | **execl(3)** | execute a file |
| **execvp()** | **execl(3)** | execute a file |
| **exit()** | **exit(3)** | terminate a process after performing cleanup |
| **exportent()** | **exportent(3)** | get exported file system information |
| **extended_to_decimal()** | **floating_to_decimal(3)** | convert floating-point value to decimal record |
| **fclose()** | **fclose(3S)** | close or flush a stream |
| **fconvert()** | **econvert(3)** | output conversion |
| **fcvt()** | **econvert(3)** | output conversion |
| **fdate()** | **fdate(3)** | return date and time in an ASCII string |
| **fdopen()** | **fopen(3S)** | open a stream |
| **feof()** | **ferror(3S)** | stream status inquiries |
| **ferror()** | **ferror(3S)** | stream status inquiries |
| **fetch()** | **dbm(3X)** | data base subroutines |
| **fflush()** | **fclose(3S)** | close or flush a stream |
| **ffs()** | **bstring(3)** | bit and byte string operations |
| **fgetc()** | **getc(3S)** | get character or integer from stream |
| **fgetgraent()** | **getgraent(3)** | get group adjunct file entry |
| **fgetgrent()** | **getgrent(3)** | get group file entry |
| **fgetpwaent()** | **getpwaent(3)** | get password adjunct file entry |
| **fgetpwent()** | **getpwent(3)** | get password file entry |
| **fgets()** | **gets(3S)** | get a string from a stream |
| **file()** | **getacinfo(3)** | get audit control file information |
| **file_to_decimal()** | **string_to_decimal(3)** | parse characters into decimal record |
| **fileno()** | **ferror(3S)** | stream status inquiries |
| **firstkey()** | **dbm(3X)** | data base subroutines |
| **floatingpoint()** | **floatingpoint(3)** | IEEE floating point definitions |
| **fopen()** | **fopen(3S)** | open a stream |
| **fprintf()** | **printf(3S)** | formatted output conversion |
| **fputc()** | **putc(3S)** | put character or word on a stream |
| **fputs()** | **puts(3S)** | put a string on a stream |
| **fread()** | **fread(3S)** | buffered binary input/output |
| **free()** | **malloc(3)** | memory allocator |
| **freopen()** | **fopen(3S)** | open a stream |
| **fscanf()** | **scanf(3S)** | formatted input conversion |
| **fseek()** | **fseek(3S)** | reposition a stream |
| **ftell()** | **fseek(3S)** | reposition a stream |
| **ftime()** | **time(3C)** | get date and time |
| **ftok()** | **ftok(3)** | standard interprocess communication package |
| **ftw()** | **ftw(3)** | walk a file tree |
| **func_to_decimal()** | **string_to_decimal(3)** | parse characters into decimal record |

| | | |
|---|---|---|
| **fwrite()** | **fread**(3S) | buffered binary input/output |
| **gcd()** | **mp**(3X) | multiple precision integer arithmetic |
| **gconvert()** | **econvert**(3) | output conversion |
| **gcvt()** | **econvert**(3) | output conversion |
| **get()** | **getacinfo**(3) | get audit control file information |
| **get_myaddress()** | **rpc**(3N) | RPC services routines |
| **getacdir()** | **getacinfo**(3) | get audit control file information |
| **getacflg()** | **getacinfo**(3) | get audit control file information |
| **getacmin()** | **getacinfo**(3) | get audit control file information |
| **getauditflagsbin()** | **getauditflags**(3) | convert audit flag specifications |
| **getauditflagschar()** | **getauditflags**(3) | convert audit flag specifications |
| **getc()** | **getc**(3S) | get character or integer from stream |
| **getchar()** | **getc**(3S) | get character or integer from stream |
| **getcwd()** | **getcwd**(3) | get pathname of current working directory |
| **getenv()** | **getenv**(3) | return value for environment name |
| **getexportent()** | **exportent**(3) | get exported file system information |
| **getexportopt()** | **exportent**(3) | get exported file system information |
| **getfauditflags()** | **getfaudflgs**(3) | generates the process audit state |
| **getfsent()** | **getfsent**(3) | get file system descriptor file entry |
| **getfsfile()** | **getfsent**(3) | get file system descriptor file entry |
| **getfsspec()** | **getfsent**(3) | get file system descriptor file entry |
| **getfstype()** | **getfsent**(3) | get file system descriptor file entry |
| **getgraent()** | **getgraent**(3) | get group adjunct file entry |
| **getgranam()** | **getgraent**(3) | get group adjunct file entry |
| **getgrent()** | **getgrent**(3) | get group file entry |
| **getgrgid()** | **getgrent**(3) | get group file entry |
| **getgrnam()** | **getgrent**(3) | get group file entry |
| **gethostbyaddr()** | **gethostent**(3N) | get network host entry |
| **gethostbyname()** | **gethostent**(3N) | get network host entry |
| **gethostent()** | **gethostent**(3N) | get network host entry |
| **getlogin()** | **getlogin**(3) | get login name |
| **getmntent()** | **getmntent**(3) | get file system descriptor file entry |
| **getnetbyaddr()** | **getnetent**(3N) | get network entry |
| **getnetbyname()** | **getnetent**(3N) | get network entry |
| **getnetent()** | **getnetent**(3N) | get network entry |
| **getnetgrent()** | **getnetgrent**(3N) | get network group entry |
| **getnetname()** | **rpc**(3N) | RPC services routines |
| **getopt()** | **getopt**(3) | get option letter from argument vector |
| **getpass()** | **getpass**(3) | read a password |
| **getprotobyname()** | **getprotoent**(3N) | get protocol entry |
| **getprotobynumber()** | **getprotoent**(3N) | get protocol entry |
| **getprotoent()** | **getprotoent**(3N) | get protocol entry |
| **getpw()** | **getpw**(3) | get name from uid |
| **getpwaent()** | **getpwaent**(3) | get password adjunct file entry |
| **getpwanam()** | **getpwaent**(3) | get password adjunct file entry |
| **getpwent()** | **getpwent**(3) | get password file entry |
| **getpwnam()** | **getpwent**(3) | get password file entry |
| **getpwuid()** | **getpwent**(3) | get password file entry |
| **getrpcbyname()** | **getrpcent**(3N) | get RPC entry |
| **getrpcbynumber()** | **getrpcent**(3N) | get RPC entry |
| **getrpcent()** | **getrpcent**(3N) | get RPC entry |
| **gets()** | **gets**(3S) | get a string from a stream |
| **getservbyname()** | **getservent**(3N) | get service entry |

| | | |
|---|---|---|
| **getservbyport()** | **getservent**(3N) | get service entry |
| **getservent()** | **getservent**(3N) | get service entry |
| **getttyent()** | **getttyent**(3) | get ttytab file entry |
| **getttynam()** | **getttyent**(3) | get ttytab file entry |
| **getusershell()** | **getusershell**(3) | get legal user shells |
| **getw()** | **getc**(3S) | get character or integer from stream |
| **getwd()** | **getwd**(3) | get current working directory pathname |
| **gmtime()** | **ctime**(3) | convert date and time |
| **grpauth()** | **pwdauth**(3) | password authentication routines |
| **gsignal()** | **ssignal**(3) | software signals |
| **gtty()** | **stty**(3C) | set and get terminal state |
| **hasmntopt()** | **getmntent**(3) | get file system descriptor file entry |
| **hcreate()** | **hsearch**(3) | manage hash search tables |
| **hdestroy()** | **hsearch**(3) | manage hash search tables |
| **host2netname()** | **rpc**(3N) | RPC services routines |
| **hsearch()** | **hsearch**(3) | manage hash search tables |
| **htonl()** | **byteorder**(3N) | convert values between host and network byte order |
| **htons()** | **byteorder**(3N) | convert values between host and network byte order |
| **index()** | **string**(3) | string operations |
| **inet_addr()** | **inet**(3N) | Internet address manipulation |
| **inet_lnaof()** | **inet**(3N) | Internet address manipulation |
| **inet_makeaddr()** | **inet**(3N) | Internet address manipulation |
| **inet_netof()** | **inet**(3N) | Internet address manipulation |
| **inet_network()** | **inet**(3N) | Internet address manipulation |
| **inet_ntoa()** | **inet**(3N) | Internet address manipulation |
| **information()** | **getacinfo**(3) | get audit control file information |
| **initgroups()** | **initgroups**(3) | initialize group access list |
| **initstate()** | **random**(3) | routines for changing random number generators |
| **innetgr()** | **getnetgrent**(3N) | get network group entry |
| **insque()** | **insque**(3) | insert/remove element from a queue |
| **isalnum()** | **ctype**(3) | character classification and conversion macros and functions |
| **isalpha()** | **ctype**(3) | character classification and conversion macros and functions |
| **isascii()** | **ctype**(3) | character classification and conversion macros and functions |
| **isatty()** | **ttyname**(3) | find name of a terminal |
| **iscntrl()** | **ctype**(3) | character classification and conversion macros and functions |
| **isdigit()** | **ctype**(3) | character classification and conversion macros and functions |
| **isgraph()** | **ctype**(3) | character classification and conversion macros and functions |
| **islower()** | **ctype**(3) | character classification and conversion macros and functions |
| **isprint()** | **ctype**(3) | character classification and conversion macros and functions |
| **ispunct()** | **ctype**(3) | character classification and conversion macros and functions |
| **issecure()** | **issecure**(3) | indicates whether system is running secure |
| **isspace()** | **ctype**(3) | character classification and conversion macros and functions |
| **isupper()** | **ctype**(3) | character classification and conversion macros and functions |
| **isxdigit()** | **ctype**(3) | character classification and conversion macros and functions |
| **itom()** | **mp**(3X) | multiple precision integer arithmetic |
| **jrand48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **key_decryptsession()** | **rpc**(3N) | RPC services routines |
| **key_encryptsession()** | **rpc**(3N) | RPC services routines |
| **key_gendes()** | **rpc**(3N) | RPC services routines |
| **key_setsecret()** | **rpc**(3N) | RPC services routines |
| **kvm_close()** | **kvm_open**(3K) | specify a kernel to examine |
| **kvm_getcmd()** | **kvm_getu**(3K) | get the u-area or invocation arguments for a process |
| **kvm_getproc()** | **kvm_nextproc**(3K) | read system process structures |

| | | |
|---|---|---|
| **kvm_getu()** | **kvm_getu**(3K) | get the u-area or invocation arguments for a process |
| **kvm_nextproc()** | **kvm_nextproc**(3K) | read system process structures |
| **kvm_nlist()** | **kvm_nlist**(3K) | obtain kernel symbol table information |
| **kvm_open()** | **kvm_open**(3K) | specify a kernel to examine |
| **kvm_read()** | **kvm_read**(3K) | copy data to or from a kernel image or running system |
| **kvm_setproc()** | **kvm_nextproc**(3K) | read system process structures |
| **kvm_write()** | **kvm_read**(3K) | copy data to or from a kernel image or running system |
| **l64a()** | **a64l**(3) | convert between long integer and base-64 ASCII string |
| **label()** | **plot**(3X) | graphics interface |
| **lcong48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **ldaclose()** | **ldclose**(3X) | close a COFF file |
| **ldahread()** | **ldahread**(3X) | read the archive header of a member of a COFF archive file |
| **ldaopen()** | **ldopen**(3X) | open a COFF file for reading |
| **ldclose()** | **ldclose**(3X) | close a COFF file |
| **ldfcn()** | **ldfcn**(3) | common object file access routines |
| **ldfhread()** | **ldfhread**(3X) | read the file header of a COFF file |
| **ldgetname()** | **ldgetname**(3X) | retrieve symbol name for COFF file symbol table entry |
| **ldlinit()** | **ldlread**(3X) | manipulate line number entries of a COFF file function |
| **ldlitem()** | **ldlread**(3X) | manipulate line number entries of a COFF file function |
| **ldlread()** | **ldlread**(3X) | manipulate line number entries of a COFF file function |
| **ldlseek()** | **ldlseek**(3X) | seek to line number entries of a section of a COFF file |
| **ldnlseek()** | **ldlseek**(3X) | seek to line number entries of a section of a COFF file |
| **ldnrseek()** | **ldrseek**(3X) | seek to relocation entries of a section of a COFF file |
| **ldnshread()** | **ldshread**(3X) | read an indexed/named section header of a COFF file |
| **ldnsseek()** | **ldsseek**(3X) | seek to an indexed/named section of a COFF file |
| **ldohseek()** | **ldohseek**(3X) | seek to the optional file header of a COFF file |
| **ldopen()** | **ldopen**(3X) | open a COFF file for reading |
| **ldrseek()** | **ldrseek**(3X) | seek to relocation entries of a section of a COFF file |
| **ldshread()** | **ldshread**(3X) | read an indexed/named section header of a COFF file |
| **ldsseek()** | **ldsseek**(3X) | seek to an indexed/named section of a COFF file |
| **ldtbindex()** | **ldtbindex**(3X) | compute the index of a symbol table entry of a COFF file |
| **ldtbread()** | **ldtbread**(3X) | read an indexed symbol table entry of a COFF file |
| **ldtbseek()** | **ldtbseek**(3X) | seek to the symbol table of a COFF file |
| **lfind()** | **lsearch**(3) | linear search and update |
| **line()** | **plot**(3X) | graphics interface |
| **linemod()** | **plot**(3X) | graphics interface |
| **localtime()** | **ctime**(3) | convert date and time |
| **lockf()** | **lockf**(3) | advisory record locking on files |
| **longjmp()** | **setjmp**(3) | non-local goto |
| **lrand48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **lsearch()** | **lsearch**(3) | linear search and update |
| **madd()** | **mp**(3X) | multiple precision integer arithmetic |
| **malloc()** | **malloc**(3) | memory allocator |
| **malloc_debug()** | **malloc**(3) | memory allocator |
| **malloc_verify()** | **malloc**(3) | memory allocator |
| **mdiv()** | **mp**(3X) | multiple precision integer arithmetic |
| **memalign()** | **malloc**(3) | memory allocator |
| **memccpy()** | **memory**(3) | memory operations |
| **memchr()** | **memory**(3) | memory operations |
| **memcmp()** | **memory**(3) | memory operations |
| **memcpy()** | **memory**(3) | memory operations |
| **memory()** | **memory**(3) | memory operations |
| **memset()** | **memory**(3) | memory operations |

| | | |
|---|---|---|
| **mfree()** | **mp**(3X) | multiple precision integer arithmetic |
| **min()** | **mp**(3X) | multiple precision integer arithmetic |
| **mkstemp()** | **mktemp**(3) | make a unique file name |
| **mktemp()** | **mktemp**(3) | make a unique file name |
| **moncontrol()** | **monitor**(3) | prepare execution profile |
| **monitor()** | **monitor**(3) | prepare execution profile |
| **monstartup()** | **monitor**(3) | prepare execution profile |
| **mout()** | **mp**(3X) | multiple precision integer arithmetic |
| **move()** | **plot**(3X) | graphics interface |
| **mrand48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **msub()** | **mp**(3X) | multiple precision integer arithmetic |
| **mtox()** | **mp**(3X) | multiple precision integer arithmetic |
| **mult()** | **mp**(3X) | multiple precision integer arithmetic |
| **netname2host()** | **rpc**(3N) | RPC services routines |
| **netname2user()** | **rpc**(3N) | RPC services routines |
| **nextkey()** | **dbm**(3X) | data base subroutines |
| **nice()** | **nice**(3C) | change priority of a process |
| **nlist()** | **nlist**(3) | get entries from name list |
| **nrand48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **ntohl()** | **byteorder**(3N) | convert values between host and network byte order |
| **ntohs()** | **byteorder**(3N) | convert values between host and network byte order |
| **on_exit()** | **on_exit**(3) | name termination handler |
| **opendir()** | **directory**(3) | directory operations |
| **openlog()** | **syslog**(3) | control system log |
| **openpl()** | **plot**(3X) | graphics interface |
| **optarg()** | **getopt**(3) | get option letter from argument vector |
| **optind()** | **getopt**(3) | get option letter from argument vector |
| **pause()** | **pause**(3C) | stop until signal |
| **pclose()** | **popen**(3S) | open or close a pipe (for I/O) from or to a process |
| **perror()** | **perror**(3) | system error messages |
| **pmap_getmaps()** | **rpc**(3N) | RPC services routines |
| **pmap_getport()** | **rpc**(3N) | RPC services routines |
| **pmap_rmtcall()** | **rpc**(3N) | RPC services routines |
| **pmap_set()** | **rpc**(3N) | RPC services routines |
| **pmap_unset()** | **rpc**(3N) | RPC services routines |
| **point()** | **plot**(3X) | graphics interface |
| **popen()** | **popen**(3S) | open or close a pipe (for I/O) from or to a process |
| **pow()** | **mp**(3X) | multiple precision integer arithmetic |
| **printf()** | **printf**(3S) | formatted output conversion |
| **prof()** | **prof**(3) | profile within a function |
| **psignal()** | **psignal**(3) | system signal messages |
| **putc()** | **putc**(3S) | put character or word on a stream |
| **putchar()** | **putc**(3S) | put character or word on a stream |
| **putenv()** | **putenv**(3) | change or add value to environment |
| **putpwent()** | **putpwent**(3) | write password file entry |
| **puts()** | **puts**(3S) | put a string on a stream |
| **putw()** | **putc**(3S) | put character or word on a stream |
| **pwdauth()** | **pwdauth**(3) | password authentication routines |
| **qsort()** | **qsort**(3) | quicker sort |
| **rand()** | **rand**(3C) | simple random number generator |
| **random()** | **random**(3) | routines for changing random number generators |
| **rcmd()** | **rcmd**(3N) | routines for returning a stream to a remote command |
| **re_comp()** | **regex**(3) | regular expression handler |

| | | |
|---|---|---|
| **re_exec()** | **regex**(3) | regular expression handler |
| **readdir()** | **directory**(3) | directory operations |
| **realloc()** | **malloc**(3) | memory allocator |
| **regex()** | **regex**(3) | regular expression handler |
| **regexp()** | **regexp**(3) | regular expression compile and match routines |
| **registerrpc()** | **rpc**(3N) | RPC services routines |
| **remexportent()** | **exportent**(3) | get exported file system information |
| **remque()** | **insque**(3) | insert/remove element from a queue |
| **res_init()** | **resolver**(3) | resolver routines |
| **res_mkquery()** | **resolver**(3) | resolver routines |
| **res_send()** | **resolver**(3) | resolver routines |
| **resolver()** | **resolver**(3) | resolver routines |
| **rewind()** | **fseek**(3S) | reposition a stream |
| **rewinddir()** | **directory**(3) | directory operations |
| **rexec()** | **rexec**(3N) | return stream to a remote command |
| **rindex()** | **string**(3) | string operations |
| **rpc_createrr()** | **rpc**(3N) | RPC services routines |
| **rpow()** | **mp**(3X) | multiple precision integer arithmetic |
| **rresvport()** | **rcmd**(3N) | routines for returning a stream to a remote command |
| **rtime()** | **rtime**(3N) | get remote time |
| **ruserok()** | **rcmd**(3N) | routines for returning a stream to a remote command |
| **scandir()** | **scandir**(3) | scan a directory |
| **scanf()** | **scanf**(3S) | formatted input conversion |
| **seconvert()** | **econvert**(3) | output conversion |
| **seed48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **seekdir()** | **directory**(3) | directory operations |
| **setac()** | **getacinfo**(3) | get audit control file information |
| **setbuf()** | **setbuf**(3S) | assign buffering to a stream |
| **setbuffer()** | **setbuf**(3S) | assign buffering to a stream |
| **setegid()** | **setuid**(3) | set user and group ID |
| **seteuid()** | **setuid**(3) | set user and group ID |
| **setexportent()** | **exportent**(3) | get exported file system information |
| **setfsent()** | **getfsent**(3) | get file system descriptor file entry |
| **setgid()** | **setuid**(3) | set user and group ID |
| **setgraent()** | **getgraent**(3) | get group adjunct file entry |
| **setgrent()** | **getgrent**(3) | get group file entry |
| **sethostent()** | **gethostent**(3N) | get network host entry |
| **setjmp()** | **setjmp**(3) | non-local goto |
| **setkey()** | **crypt**(3) | password and data encryption |
| **setlinebuf()** | **setbuf**(3S) | assign buffering to a stream |
| **setlogmask()** | **syslog**(3) | control system log |
| **setmntent()** | **getmntent**(3) | get file system descriptor file entry |
| **setnetent()** | **getnetent**(3N) | get network entry |
| **setnetgrent()** | **getnetgrent**(3N) | get network group entry |
| **setprotoent()** | **getprotoent**(3N) | get protocol entry |
| **setpwaent()** | **getpwaent**(3) | get password adjunct file entry |
| **setpwent()** | **getpwent**(3) | get password file entry |
| **setpwfile()** | **getpwent**(3) | get password file entry |
| **setrgid()** | **setuid**(3) | set user and group ID |
| **setruid()** | **setuid**(3) | set user and group ID |
| **setservent()** | **getservent**(3N) | get service entry |
| **setstate()** | **random**(3) | routines for changing random number generators |
| **setttyent()** | **getttyent**(3) | get ttytab file entry |

| | | |
|---|---|---|
| **setuid()** | **setuid**(3) | set user and group ID |
| **setusershell()** | **getusershell**(3) | get legal user shells |
| **setvbuf()** | **setbuf**(3S) | assign buffering to a stream |
| **sfconvert()** | **econvert**(3) | output conversion |
| **sgconvert()** | **econvert**(3) | output conversion |
| **sigfpe()** | **sigfpe**(3) | signal handling for specific SIGFPE codes |
| **siginterrupt()** | **siginterrupt**(3) | allow signals to interrupt system calls |
| **signal()** | **signal**(3) | simplified software signal facilities |
| **single_to_decimal()** | **floating_to_decimal**(3) | convert floating-point value to decimal record |
| **sleep()** | **sleep**(3) | suspend execution for interval |
| **space()** | **plot**(3X) | graphics interface |
| **sprintf()** | **printf**(3S) | formatted output conversion |
| **srand()** | **rand**(3C) | simple random number generator |
| **srand48()** | **drand48**(3) | generate uniformly distributed pseudo-random numbers |
| **srandom()** | **random**(3) | routines for changing random number generators |
| **sscanf()** | **scanf**(3S) | formatted input conversion |
| **ssignal()** | **ssignal**(3) | software signals |
| **store()** | **dbm**(3X) | data base subroutines |
| **strcat()** | **string**(3) | string operations |
| **strchr()** | **string**(3) | string operations |
| **strcmp()** | **string**(3) | string operations |
| **strcpy()** | **string**(3) | string operations |
| **strcspn()** | **string**(3) | string operations |
| **strdup()** | **string**(3) | string operations |
| **string_to_decimal()**(3) | **string_to_decimal**(3) | parse characters into decimal record |
| **strlen()** | **string**(3) | string operations |
| **strncat()** | **string**(3) | string operations |
| **strncmp()** | **string**(3) | string operations |
| **strncpy()** | **string**(3) | string operations |
| **strpbrk()** | **string**(3) | string operations |
| **strrchr()** | **string**(3) | string operations |
| **strspn()** | **string**(3) | string operations |
| **strtod()** | **strtod**(3) | convert string to double-precision number |
| **strtok()** | **string**(3) | string operations |
| **strtol()** | **strtol**(3) | convert string to integer |
| **stty()** | **stty**(3C) | set and get terminal state |
| **svc_destroy()** | **rpc**(3N) | RPC services routines |
| **svc_fds()** | **rpc**(3N) | RPC services routines |
| **svc_freeargs()** | **rpc**(3N) | RPC services routines |
| **svc_getargs()** | **rpc**(3N) | RPC services routines |
| **svc_getcaller()** | **rpc**(3N) | RPC services routines |
| **svc_getreq()** | **rpc**(3N) | RPC services routines |
| **svc_register()** | **rpc**(3N) | RPC services routines |
| **svc_run()** | **rpc**(3N) | RPC services routines |
| **svc_sendreply()** | **rpc**(3N) | RPC services routines |
| **svc_unregister()** | **rpc**(3N) | RPC services routines |
| **svcerr_auth()** | **rpc**(3N) | RPC services routines |
| **svcerr_decode()** | **rpc**(3N) | RPC services routines |
| **svcerr_noproc()** | **rpc**(3N) | RPC services routines |
| **svcerr_noprog()** | **rpc**(3N) | RPC services routines |
| **svcerr_progvers()** | **rpc**(3N) | RPC services routines |
| **svcerr_systemerr()** | **rpc**(3N) | RPC services routines |
| **svcerr_weakauth()** | **rpc**(3N) | RPC services routines |

| | | |
|---|---|---|
| **svcfd_create()** | **rpc**(3N) | RPC services routines |
| **svcraw_create()** | **rpc**(3N) | RPC services routines |
| **svctcp_create()** | **rpc**(3N) | RPC services routines |
| **svcudp_create()** | **rpc**(3N) | RPC services routines |
| **swab()** | **swab**(3) | swap bytes |
| **sys_errlist()** | **perror**(3) | system error messages |
| **sys_nerr()** | **perror**(3) | system error messages |
| **sys_siglist()** | **psignal**(3) | system signal messages |
| **syslog()** | **syslog**(3) | control system log |
| **system()** | **system**(3) | issue a shell command |
| **tdelete()** | **tsearch**(3) | manage binary search trees |
| **telldir()** | **directory**(3) | directory operations |
| **tempnam()** | **tmpnam**(3S) | create a name for a temporary file |
| **tfind()** | **tsearch**(3) | manage binary search trees |
| **tgetent()** | **termcap**(3X) | terminal independent operation routines |
| **tgetflag()** | **termcap**(3X) | terminal independent operation routines |
| **tgetnum()** | **termcap**(3X) | terminal independent operation routines |
| **tgetstr()** | **termcap**(3X) | terminal independent operation routines |
| **tgoto()** | **termcap**(3X) | terminal independent operation routines |
| **time()** | **time**(3C) | get date and time |
| **timegm()** | **ctime**(3) | convert date and time |
| **timelocal()** | **ctime**(3) | convert date and time |
| **times()** | **times**(3C) | get process times |
| **timezone()** | **timezone**(3C) | get time zone name given offset from GMT |
| **tmpfile()** | **tmpfile**(3S) | create a temporary file |
| **tmpnam()** | **tmpnam**(3S) | create a name for a temporary file |
| **toascii()** | **ctype**(3) | character classification and conversion macros and functions |
| **tolower()** | **ctype**(3) | character classification and conversion macros and functions |
| **toupper()** | **ctype**(3) | character classification and conversion macros and functions |
| **tputs()** | **termcap**(3X) | terminal independent operation routines |
| **tsearch()** | **tsearch**(3) | manage binary search trees |
| **ttyname()** | **ttyname**(3) | find name of a terminal |
| **ttyslot()** | **ttyslot**(3) | find the slot in the utmp file of the current process |
| **twalk()** | **tsearch**(3) | manage binary search trees |
| **tzset()** | **ctime**(3) | convert date and time |
| **tzsetwall()** | **ctime**(3) | convert date and time |
| **ualarm()** | **ualarm**(3) | schedule signal after interval in microseconds |
| **ulimit()** | **ulimit**(3C) | get and set user limits |
| **ungetc()** | **ungetc**(3S) | push character back into input stream |
| **user2netname()** | **rpc**(3N) | RPC services routines |
| **usleep()** | **usleep**(3) | suspend execution for interval in microseconds |
| **utime()** | **utime**(3C) | set file times |
| **valloc()** | **malloc**(3) | memory allocator |
| **varargs()** | **varargs**(3) | handle variable argument list |
| **vfprintf()** | **vprintf**(3S) | print formatted output of a varargs argument list |
| **vlimit()** | **vlimit**(3C) | control maximum system resource consumption |
| **vprintf()** | **vprintf**(3S) | print formatted output of a varargs argument list |
| **vsprintf()** | **vprintf**(3S) | print formatted output of a varargs argument list |
| **vtimes()** | **vtimes**(3C) | get information about resource utilization |
| **xdr_accepted_reply()** | **xdr**(3N) | XDR functions |
| **xdr_array()** | **xdr**(3N) | XDR functions |
| **xdr_authunix_parms()** | **xdr**(3N) | XDR functions |
| **xdr_bool()** | **xdr**(3N) | XDR functions |

| | | |
|---|---|---|
| **xdr_bytes()** | **xdr**(3N) | XDR functions |
| **xdr_callhdr()** | **xdr**(3N) | XDR functions |
| **xdr_callmsg()** | **xdr**(3N) | XDR functions |
| **xdr_char()** | **xdr**(3N) | XDR functions |
| **xdr_destroy()** | **xdr**(3N) | XDR functions |
| **xdr_double()** | **xdr**(3N) | XDR functions |
| **xdr_enum()** | **xdr**(3N) | XDR functions |
| **xdr_float()** | **xdr**(3N) | XDR functions |
| **xdr_getpos()** | **xdr**(3N) | XDR functions |
| **xdr_inline()** | **xdr**(3N) | XDR functions |
| **xtom()** | **mp**(3X) | multiple precision integer arithmetic |
| **yp_all()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_bind()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_first()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_get_default_domain()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_master()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_match()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_next()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_order()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_unbind()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yp_update()** | **ypupdate**(3N) | update YP information |
| **ypclnt()** | **ypclnt**(3N) | Yellow Pages client interface |
| **yperr_string()** | **ypclnt**(3N) | Yellow Pages client interface |
| **ypprot_err()** | **ypclnt**(3N) | Yellow Pages client interface |

NAME

a64l, l64a – convert between long integer and base-64 ASCII string

SYNOPSIS

**long a64l(s)**
**char *s;**

**char *l64a(l)**
**long l;**

DESCRIPTION

These functions are used to maintain numbers stored in *base-64* ASCII characters. This is a notation by which long integers can be represented by up to six characters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are '.' for 0, '/' for 1, **0** through **9** for 2–11, **A** through **Z** for 12–37, and **a** through **z** for 38–63.

**a64l**( ) takes a pointer to a NULL-terminated base-64 representation and returns a corresponding **long** value. If the string pointed to by *s* contains more than six characters, **a64l**( ) will use the first six.

**l64a**( ) takes a **long** argument and returns a pointer to the corresponding base-64 representation. If the argument is 0, **l64a**( ) returns a pointer to a NULL string.

BUGS

The value returned by **l64a**( ) is a pointer into a static buffer, the contents of which are overwritten by each call.

**NAME**

abort – generate a fault

**SYNOPSIS**

**abort()**

**DESCRIPTION**

**abort()** first closes all open files if possible, then sends an IOT signal to the process. This signal usually results in termination with a core dump, which may be used for debugging.

It is possible for **abort()** to return control if **SIGIOT** is caught or ignored, in which case the value returned is that of the **kill**(2V) system call.

**SEE ALSO**

**adb**(1), **exit**(2), **kill**(2V), **signal**(3)

**DIAGNOSTICS**

If **SIGIOT** is neither caught nor ignored, and the current directory is writable, a core dump is produced and the message '**abort – core dumped**' is written by the shell.

NAME

abs – integer absolute value

SYNOPSIS

**abs(i)**
**int i;**

DESCRIPTION

**abs( )** returns the absolute value of its integer operand.

SEE ALSO

**rint**(3M) for **fabs( )**

BUGS

Applying the **abs( )** function to the most negative integer generates a result which is the most negative integer. That is, **abs(0x80000000)** returns **0x80000000** as a result.

NAME

alarm – schedule signal after specified time

SYNOPSIS

**alarm(seconds)**
**unsigned seconds;**

DESCRIPTION

**alarm()** sends signal SIGALRM, see **sigvec**(2), to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is canceled. Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 2147483647 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

**sigpause**(2), **sigvec**(2), **signal**(3), **sleep**(3), **ualarm**(3), **usleep**(3)

NAME

assert – program verification

SYNOPSIS

**#include <assert.h>**

**assert(expression)**

DESCRIPTION

**assert( )** is a macro that indicates *expression* is expected to be true at this point in the program. It exits (see **exit**(2) with a diagnostic comment on the standard output when *expression* is false (0). Compiling with the **cc**(1V) option **–DNDEBUG** effectively deletes **assert( )** from the program.

SEE ALSO

**cc**(1V) **exit**(2)

DIAGNOSTICS

**Assertion failed: file** *f* **line** *n*

*f* is the source file and *n* the source line number of the **assert( )** statement.

NAME

audit_args, audit_text – produce text audit message

SYNOPSIS

```
#include <sys/label.h>
#include <sys/audit.h>

audit_args(event, argc, argv)
int event;
int argc;
char **argv;

audit_text(event, error, retval, argc, argv)
int event;
int error;
int retval;
int argc;
char **argv;
```

DESCRIPTION

These functions provide text interfaces to the **audit**(2) system call. In both calls, the *event* parameter identifies the event class of the action, and *argc* is the number of strings found in the vector *argv*. The **error** parameter is used to determine the failure or success of the audited operation. A negative value is always audited. A zero value is audited as a successful event. A positive value is audited as an event failure. The *retval* parameter is the return value or exit code that the invoking program will have.

**audit_args**() is equivalent to **audit_text**() with **error** and *retval* parameters of –1.

SEE ALSO

**audit**(2)

NAME

bindresvport – bind a socket to a privileged IP port

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>

int bindresvport(sd, sin)
int sd;
struct sockaddr_in *sin;
```

DESCRIPTION

**bindresvport( )** is used to bind a socket descriptor to a privileged IP port, that is, a port number in the range 0-1023. The routine returns 0 if it is successful, otherwise −1 is returned and **errno** set to reflect the cause of the error. This routine differs with **rresvport** (see **rcmd**(3N)) in that this works for any IP socket, whereas **rresvport( )** only works for TCP.

Only root can bind to a privileged port; this call will fail for any other users.

SEE ALSO

**rcmd**(3N)

## NAME

bsearch – binary search a sorted table

## SYNOPSIS

**#include <search.h>**

**char *bsearch ((char *) key, (char *) base, nel, sizeof (*key), compar)**
**unsigned nel;**
**int (*compar)( );**

## DESCRIPTION

**bsearch( )** is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order according to a provided comparison function. *key* points to a datum instance to be sought in the table. *base* points to the element at the base of the table. *nel* is the number of elements in the table. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero as accordingly the first argument is to be considered less than, equal to, or greater than the second.

## EXAMPLE

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

This code fragment reads in strings and either finds the corresponding node, in which case it prints out the string and its length, or it prints an error message.

```
#include <stdio.h>
#include <search.h>
#define  TABSIZE        1000
struct node {                   /* these are stored in the table */
        char *string;
        int length;
};
struct node table[TABSIZE];     /* table to be searched */
        .
        .
        .
{
        struct node *node_ptr, node;
        int node_compare( ); /* routine to compare 2 nodes */
        char str_space[20];  /* space to read string into */
        .
        .
        .
        node.string = str_space;
        while (scanf("%s", node.string) != EOF) {
                node_ptr = (struct node *)bsearch((char *)(&node),
                        (char *)table, TABSIZE,
                        sizeof(struct node), node_compare);
                if (node_ptr != NULL) {
                        (void)printf("string = %20s, length = %d\n",
                                node_ptr->string, node_ptr->length);
                } else {
                        (void)printf("not found: %s\n", node.string);
                }
        }
}
/*
        This routine compares two nodes based on an
        alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
struct node *node1, *node2;
{
        return strcmp(node1->string, node2->string);
}
```

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

SEE ALSO

**hsearch**(3), **lsearch**(3), **qsort**(3), **tsearch**(3)

**DIAGNOSTICS**

A NULL pointer is returned if the key cannot be found in the table.

NAME

bstring, bcopy, bcmp, bzero, ffs – bit and byte string operations

SYNOPSIS

```
bcopy(b1, b2, length)
char *b1, *b2;
int length;

int bcmp(b1, b2, length)
char *b1, *b2;
int length;

bzero(b, length)
char *b;
int length;

int ffs(i)
int i;
```

DESCRIPTION

The functions **bcopy**, **bcmp**, and **bzero**( ) operate on variable length strings of bytes. They do not check for NULL bytes as the routines in **string**(3) do.

**bcopy**( ) copies *length* bytes from string *b1* to the string *b2*. Overlapping strings are handled correctly.

**bcmp**( ) compares byte string *b1* against byte string *b2*, returning zero if they are identical, non-zero otherwise. Both strings are assumed to be *length* bytes long. **bcmp**( ) of length zero bytes always returns zero.

*bzero* places *length* 0 bytes in the string *b*.

*ffs* finds the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1 from the right. A return value of zero indicates that the value passed is zero.

CAVEAT

The **bcmp**( ) and **bcopy**( ) routines take parameters backwards from **strcmp**( ) and **strcpy**.

SEE ALSO

**string**(3)

NAME

byteorder, htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>

netlong = htonl(hostlong);
u_long netlong, hostlong;

netshort = htons(hostshort);
u_short netshort, hostshort;

hostlong = ntohl(netlong);
u_long hostlong, netlong;

hostshort = ntohs(netshort);
u_short hostshort, netshort;
```

DESCRIPTION

These routines convert 16 and 32 bit quantities between network byte order and host byte order. On Sun-2, Sun-3 and Sun-4 systems, these routines are defined as NULL macros in the include file **<netinet/in.h>**. On Sun386i systems, these routines are functional since its host byte order is different from network byte order.

These routines are most often used in conjunction with Internet addresses and ports as returned by **gethostent**(3N) and **getservent**(3N).

SEE ALSO

**gethostent**(3N), **getservent**(3N)

NAME

clock – report CPU time used

SYNOPSIS

**long clock ( )**

DESCRIPTION

**clock( )** returns the amount of CPU time (in microseconds) used since the first call to **clock**. The time reported is the sum of the user and system times of the calling process and its terminated child processes for which it has executed **wait**(2) or **system**(3).

The resolution of the clock is 16.667 milliseconds.

SEE ALSO

**wait**(2), **system**(3), **times**(3C), **times**(3V)

BUGS

The value returned by **clock( )** is defined in microseconds for compatibility with systems that have CPU clocks with much higher resolution. Because of this, the value returned will wrap around after accumulating only 2147 seconds of CPU time (about 36 minutes).

NAME

crypt, _crypt, setkey, encrypt – password and data encryption

SYNOPSIS

**char *crypt(key, salt)**
**char *key, *salt;**

**char *_crypt(key, salt)**
**char *key, *salt;**

**setkey(key)**
**char *key;**

**encrypt(block, edflag)**
**char *block;**

DESCRIPTION

**crypt( )** is the password encryption routine, based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to **crypt( )** is normally a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. Unless it starts with '##' or '#$', the *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

If the *salt* string starts with '##', **pwdauth**(3) is called. If *pwdauth* returns TRUE, the salt is returned from **crypt.** Otherwise, NULL is returned. If the *salt* string starts with '#$', **grpauth** (see **pwaudth**(3)) is called. If **grpauth** returns TRUE, the salt is returned from **crypt**. Otherwise, NULL is returned. If there is a valid reason not to have this authentication happen, calling *_crypt* avoids authentication.

The *setkey* and *encrypt* entries provide (rather primitive) access to the DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is set into the machine. This is the key that will be used with the above mentioned algorithm to encrypt or decrypt the string *block* with the function *encrypt*.

The argument to the *encrypt* entry is a character array of length 64 containing only the characters with numerical value 0 and 1. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is zero, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

**login**(1), **passwd**(1), **getpass**(3), **pwdauth**(3), **passwd**(5)

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

ctermid – generate filename for terminal

SYNOPSIS

**#include <stdio.h>**
**char *ctermid (s)**
**char *s;**

DESCRIPTION

**ctermid( )** generates the pathname of the controlling terminal for the current process, and stores it in a string.

If *s* is a NULL pointer, the string is stored in an internal static area, the contents of which are overwritten at the next call to **ctermid**, and the address of which is returned. Otherwise, *s* is assumed to point to a character array of at least **l_ctermid** elements; the path name is placed in this array and the value of *s* is returned. The constant **l_ctermid** is defined in the **<stdio.h>** header file.

NOTES

The difference between **ctermid( )** and **ttyname**(3) is that **ttyname( )** must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while **ctermid( )** returns a string (**/dev/tty**) that will refer to the terminal if used as a file name. Thus **ttyname( )** is useful only if the process already has at least one file open to a terminal. **ctermid( )** is useful largely for making code portable to (non-UNIX) systems where the current terminal is referred to by a name other than **/dev/tty**.

SEE ALSO

**ttyname**(3)

## NAME

ctime, localtime, gmtime, asctime, dysize, timelocal, timegm, tzset, tzsetwall – convert date and time

## SYNOPSIS

```
#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *ctime(clock)
long *clock;

int dysize(y)
int y;

time_t timelocal(tm)
struct tm *tm;

time_t timegm(tm)
struct tm *tm;

void tzset()

void tzsetwall()
```

## DESCRIPTION

**localtime**( ) and **gmtime**( ) return pointers to structures containing the time, broken down into various components of that time represented in a particular time zone. **localtime**( ) breaks down a time specified by the **clock**( ) argument, correcting for the time zone and any time zone adjustments (such as Daylight Savings Time). Before doing so, **localtime**( ) calls *tzset* (if *tzset* has not been called in the current process). **gmtime**( ) breaks down a time specified by the **clock**( ) argument into GMT, which is the time the system uses.

*asctime* converts a time value contained in a "tm" structure to a 26-character string of the form:

**Sun Sep 16 01:03:52 1973\n\0**

Each field has a constant width. *asctime* returns a pointer to the string.

**ctime**( ) converts a long integer, pointed to by *clock*, to a 26-character string of the form produced by *asctime*. It first breaks down **clock**( ) to a **struct tm** by calling **localtime**( ), and then calls *asctime* to convert that **struct tm** to a string.

*dysize* returns the number of days in the argument year, either 365 or 366.

**timelocal**( ) and **timegm**( ) convert the time specified by the **tm** argument to a time value that represents that time expressed as the number of seconds since Jan. 1, 1970, 00:00, Greenwich Mean Time. **timelocal**( ) converts a **struct tm** that represents local time, correcting for the time zone and any time zone adjustments (such as Daylight Savings Time). Before doing so, **timelocal**( ) calls *tzset* (if *tzset* has not been called in the current process). **timegm**( ) converts a **struct tm** that represents GMT.

*tzset* uses the value of the environment variable TZ to set time conversion information used by **localtime**( ). If TZ is absent from the environment, the best available approximation to local wall clock time is used by **localtime**( ). If TZ appears in the environment but its value is a NULL string, Greenwich Mean Time is used; if TZ appears and begins with a slash, it is used as the absolute pathname of the *tzfile-format* (see **tzfile**(5)) file from which to read the time conversion information; if TZ appears and begins with a character other than a slash, it is used as a pathname relative to a system time conversion information directory.

*tzsetwall* sets things up so that **localtime()** returns the best available approximation of local wall clock time.

Declarations of all the functions and externals, and the "tm" structure, are in the **<time.h>** header file. The structure (of type) **struct tm** includes the following fields:

```
int tm_sec;     /* seconds (0 - 59) */
int tm_min;     /* minutes (0 - 59) */
int tm_hour;    /* hours (0 - 23) */
int tm_mday;    /* day of month (1 - 31) */
int tm_mon;     /* month of year (0 - 11) */
int tm_year;    /* year - 1900 */
int tm_wday;    /* day of week (Sunday = 0) */
int tm_yday;    /* day of year (0 - 365) */
int tm_isdst;   /* 1 if DST in effect */
char *tm_zone;  /* abbreviation of timezone name */
long tm_gmtoff; /* offset from GMT in seconds */
```

**tm_isdst** is non-zero if Daylight Savings Time is in effect. **tm_zone** points to a string that is the name used for the local time zone at the time being converted. **tm_gmtoff** is the offset (in seconds) of the time represented from GMT, with positive values indicating East of Greenwich.

FILES

**/usr/share/lib/zoneinfo** standard time conversion information directory
**/usr/share/lib/zoneinfo/localtime**
local time zone file

SEE ALSO

**gettimeofday**(2), **ctime**(3V), **getenv**(3), **time**(3C), **environ**(5V), **tzfile**(5)

BUGS

The return values point to static data, whose contents are overwritten by each call. The **tm_zone** field of a returned **struct tm** points to a static array of characters, which will also be overwritten at the next call (and by calls to *tzset* or *tzsetwall*).

NAME

ctype, isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii, isgraph, toupper, tolower, toascii – character classification and conversion macros and functions

SYNOPSIS

**#include <ctype.h>**

**isalpha(c)**

...

CHARACTER CLASSIFICATION MACROS

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *isascii* is defined on all integer values; the rest are defined only where **isascii**(*c*) is true and on the single non-ASCII value EOF (see **stdio**(3S)).

| | |
|---|---|
| **isalpha**(*c*) | *c* is a letter |
| **isupper**(*c*) | *c* is an upper case letter |
| **islower**(*c*) | *c* is a lower case letter |
| **isdigit**(*c*) | *c* is a digit [0-9]. |
| **isxdigit**(*c*) | *c* is a hexadecimal digit [0-9], [A-F], or [a-f]. |
| **isalnum**(*c*) | *c* is an alphanumeric character, that is, *c* is a letter or a digit |
| **isspace**(*c*) | *c* is a space, tab, carriage return, newline, vertical tab, or formfeed |
| **ispunct**(*c*) | *c* is a punctuation character (neither control nor alphanumeric) |
| **isprint**(*c*) | *c* is a printing character, code 040(8) (space) through 0176 (tilde) |
| **iscntrl**(*c*) | *c* is a delete character (0177) or ordinary control character (less than 040). |
| **isascii**(*c*) | *c* is an ASCII character, code less than 0200 |
| **isgraph**(*c*) | *c* is a visible graphic character, code 041 (exclamation mark) through 0176 (tilde). |

CHARACTER CONVERSION MACROS

These macros perform simple conversions on single characters.

| | |
|---|---|
| **toupper**(*c*) | converts *c* to its upper-case equivalent. Note: this *only* works where *c* is known to be a lower-case character to start with (presumably checked using *islower*). |
| **tolower**(*c*) | converts *c* to its lower-case equivalent. Note: this *only* works where *c* is known to be a upper-case character to start with (presumably checked using *isupper*). |
| **toascii**(*c*) | masks *c* with the correct value so that *c* is guaranteed to be an ASCII character in the range 0 through 0x7f. |

DIAGNOSTICS

If the argument to any of these macros is not in the domain of the function, the result is undefined.

SEE ALSO

**ctype**(3V), **stdio**(3S), **ascii**(7)

**NAME**

curses – cursor addressing and screen display library

**SYNOPSIS**

**cc** [ *flags* ] *files* **–lcurses –ltermcap** [ *libraries* ]

**DESCRIPTION**

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the **refresh( )** tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine **initscr( )** must be called before any of the other routines that deal with windows and screens are used. The routine **endwin( )** should be called before exiting.

**SEE ALSO**

**tic**(8V), **ioctl**(2), **curses**(3V), **getenv**(3), **tty**(4), **terminfo**(5V), **term**(5V), **termcap**(5)

*Programming Utilities and Libraries*

**Curses Functions**

| | |
|---|---|
| **addch**(*ch*) | add a character to *stdscr* |
| **addstr**(*str*) | add a string to *stdscr* |
| **box**(*win,vert,hor*) | draw a box around a window |
| **cbreak( )** | set cbreak mode |
| **clear( )** | clear *stdscr* |
| **clearok**(*scr,boolf*) | set clear flag for *scr* |
| **clrtobot( )** | clear to bottom on *stdscr* |
| **clrtoeol( )** | clear to end of line on *stdscr* |
| **delch( )** | delete a character |
| **deleteln( )** | delete a line |
| **delwin**(*win*) | delete *win* |
| **echo( )** | set echo mode |
| **endwin( )** | end window modes |
| **erase( )** | erase *stdscr* |
| **flusok**(*win,boolf*) | set flush-on-refresh flag for *win* |
| **getch( )** | get a char through *stdscr* |
| **getcap**(*name*) | get terminal capability *name* |
| **getstr**(*str*) | get a string through *stdscr* |
| **gettmode( )** | get tty modes |
| **getyx**(*win,y,x*) | get (*y,x*) co-ordinates |
| **inch( )** | get char at current (*y,x*) co-ordinates |
| **initscr( )** | initialize screens |
| **insch**(*c*) | insert a char |
| **insertln( )** | insert a line |
| **leaveok**(*win,boolf*) | set leave flag for *win* |
| **longname**(*termbuf,name*) | get long name from *termbuf* |
| **move**(*y,x*) | move to (*y,x*) on *stdscr* |
| **mvcur**(*lasty,lastx,newy,newx*) | actually move cursor |
| **newwin**(*lines,cols,begin_y,begin_x*) | create a new window |
| **nl( )** | set NEWLINE mapping |
| **nocbreak( )** | unset cbreak mode |
| **noecho( )** | unset echo mode |
| **nonl( )** | unset NEWLINE mapping |
| **noraw( )** | unset raw mode |
| **overlay**(*win1,win2*) | overlay *win1* on *win2* |
| **overwrite**(*win1,win2*) | overwrite *win1* on top of *win2* |
| **printw**(*fmt,arg1,arg2,...*) | printf on *stdscr* |
| **raw( )** | set raw mode |

| | |
|---|---|
| **refresh()** | make current screen look like *stdscr* |
| **resetty()** | reset tty flags to stored value |
| **savetty()** | stored current tty flags |
| **scanw**(*fmt,arg1,arg2,...*) | scanf through *stdscr* |
| **scroll**(*win*) | scroll *win* one line |
| **scrollok**(*win,boolf*) | set scroll flag |
| **setterm**(*name*) | set term variables for name |
| **standend()** | end standout mode |
| **standout()** | start standout mode |
| **subwin**(*win,lines,cols,begin_y,begin_x*) | create a subwindow |
| **touchline**(*win,y,sx,ex*) | mark line *y* *sx* through *sy* as changed |
| **touchoverlap**(*win1,win2*) | mark overlap of *win1* on *win2* as changed |
| **touchwin**(*win*) | "change" all of *win* |
| **unctrl**(*ch*) | printable version of *ch* |
| **waddch**(*win,ch*) | add char to *win* |
| **waddstr**(*win,str*) | add string to *win* |
| **wclear**(*win*) | clear *win* |
| **wclrtobot**(*win*) | clear to bottom of *win* |
| **wclrtoeol**(*win*) | clear to end of line on *win* |
| **wdelch** (*win,c*) | delete char from *win* |
| **wdeleteln**(*win*) | delete line from *win* |
| **werase**(*win*) | erase *win* |
| **wgetch**(*win*) | get a char through *win* |
| **wgetstr** (*win,str*) | get a string through *win* |
| **winch**(*win*) | get char at current (*y,x*) in *win* |
| **winsch** (*win,c*) | insert character into *win* |
| **winsertln**(*win*) | insert line into *win* |
| **wmove**(*win,y,x*) | set current (*y,x*) co-ordinates on *win* |
| **wprintw**(*win,fmt,arg1,arg2,...*) | printf on *win* |
| **wrefresh**(*win*) | make screen look like *win* |
| **wscanw**(*win,fmt,arg1,arg2,...*) | scanf through *win* |
| **wstandend**(*win*) | end standout mode on *win* |
| **wstandout**(*win*) | start standout mode on *win* |

NAME

cuserid – get character login name of the user

SYNOPSIS

**#include <stdio.h>**

**char *cuserid (s)**
**char *s;**

DESCRIPTION

**cuserid( )** generates a character-string representation of the login name that the owner of the current process is logged in under. If *s* is a NULL pointer, this representation is generated in an internal static area, the address of which is returned. Otherwise, *s* is assumed to point to an array of at least **l_cuserid** characters; the representation is left in this array. The constant **l_cuserid** is defined in the **<stdio.h>** header file.

SEE ALSO

**getlogin**(3), **getpwent**(3)

DIAGNOSTICS

If the login name cannot be found, **cuserid( )** returns a NULL pointer; if *s* is not a NULL pointer, a NULL character ('\0') will be placed at s[0].

NAME

dbm, dbminit, dbmclose, fetch, store, delete, firstkey, nextkey – data base subroutines

SYNOPSIS

```
#include <dbm.h>

typedef struct {
        char *dptr;
        int dsize;
} datum;

dbminit(file)
char *file;

dbmclose()

datum fetch(key)
datum key;

store(key, content)
datum key, content;

delete(key)
datum key;

datum firstkey()

datum nextkey(key)
datum key;
```

DESCRIPTION

Note: the **dbm()** library has been superceded by **ndbm**(3), and is now implemented using **ndbm**.

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. The functions are obtained with the loader option **–ldbm**.

*key*s and *content*s are described by the **datum** typedef. A **datum** specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

Before a database can be accessed, it must be opened by **dbminit**. At the time of this call, the files *file***.dir** and *file***.pag** must exist. (An empty database is created by creating zero-length **.dir** and **.pag** files.)

A database may be closed by calling **dbmclose**. You must close a database before opening a new one.

Once open, the data stored under a key is accessed by **fetch()** and data is placed under a key by **store**. A key (and its associated contents) is deleted by **delete**. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of **firstkey()** and **nextkey**. **firstkey()** will return the first key in the database. With any key **nextkey()** will return the next key in the database. This code will traverse the data base:

```
for (key = firstkey(); key.dptr != NULL; key = nextkey(key))
```

SEE ALSO

**ndbm**(3)

DIAGNOSTICS

All functions that return an **int** indicate errors with negative values. A zero return indicates no error. Routines that return a **datum** indicate errors with a NULL (**0**) *dptr*.

BUGS

The **.pag** file will contain holes so that its apparent size is about four times its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files

cannot be copied by normal means (**cp**(1), **cat**(1V), **tp**(5), **tar**(1), **ar**(1)) without filling in the holes.

*dptr* pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. **store()** will return an error in the event that a disk block fills with inseparable data.

**delete()** does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by **firstkey()** and **nextkey()** depends on a hashing function, not on anything interesting.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

NAME

decimal_to_single, decimal_to_double, decimal_to_extended – convert decimal record to floating-point value

SYNOPSIS

```
#include <floatingpoint.h>

void decimal_to_single(px, pm, pd, ps)
single *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void decimal_to_double(px, pm, pd, ps)
double *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void decimal_to_extended(px, pm, pd, ps)
extended *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;
```

DESCRIPTION

The **decimal_to_floating()** functions convert the decimal record at **pd* into a floating-point value at **px*, observing the modes specified in **pm* and setting exceptions in **ps*. If there are no IEEE exceptions, **ps* will be zero.

*pd->sign* and *pd->fpclass* are always taken into account. *pd->exponent* and *pd->ds* are used when *pd->fpclass* is *fp_normal* or *fp_subnormal*. In these cases *pd->ds* must contain one or more ascii digits followed by a NULL. **px* is set to a correctly rounded approximation to

**(pd->sign)*(pd->ds)*10**(pd->exponent)**

Thus if *pd->exponent* == –2 and *pd->ds* == "1234", **px* will get 12.34 rounded to storage precision. *pd->ds* cannot have more than **DECIMAL_STRING_LENGTH-1** significant digits because one character is used to terminate the string with a NULL. If *pd->more != 0* on input then additional nonzero digits follow those in *pd->ds; fp_inexact* is set accordingly on output in **ps*.

**px* is correctly rounded according to the IEEE rounding modes in *pm->rd*. **ps* is set to contain *fp_inexact*, *fp_underflow*, or *fp_overflow* if any of these arise.

*pd->ndigits*, *pm->df*, and *pm->ndigits* are not used.

**strtod**(3), **scanf**(3), **fscanf**(3), and **sscanf**(3) all use **decimal_to_double**.

SEE ALSO

**scanf**(3S), **scanf**(3V), **strtod**(3)

NAME

des_crypt, ecb_crypt, cbc_crypt, des_setparity – fast DES encryption

SYNOPSIS

```
#include <des_crypt.h>

int ecb_crypt(key, data, datalen, mode)
char *key;
char *data;
unsigned datalen;
unsigned mode;

int cbc_crypt(key, data, datalen, mode, ivec)
char *key;
char *data;
unsigned datalen;
unsigned mode;
char *ivec;

void des_setparity(key)
char *key;
```

DESCRIPTION

**ecb_crypt()** and **cbc_crypt()** implement the NBS DES (Data Encryption Standard). These routines are faster and more general purpose than **crypt**(3). They also are able to utilize DES hardware if it is available. **ecb_crypt()** encrypts in ECB (Electronic Code Book) mode, which encrypts blocks of data independently. **cbc_crypt()** encrypts in CBC (Cipher Block Chaining) mode, which chains together successive blocks. CBC mode protects against insertions, deletions and substitutions of blocks. Also, regularities in the clear text will not appear in the cipher text.

Here is how to use these routines. The first parameter, *key*, is the 8-byte encryption key with parity. To set the key's parity, which for DES is in the low bit of each byte, use *des_setparity*. The second parameter, *data*, contains the data to be encrypted or decrypted. The third parameter, *datalen*, is the length in bytes of *data*, which must be a multiple of 8. The fourth parameter, *mode*, is formed by OR'ing together some things. For the encryption direction 'or' in either DES_ENCRYPT or DES_DECRYPT. For software versus hardware encryption, 'or' in either DES_HW or DES_SW. If DES_HW is specified, and there is no hardware, then the encryption is performed in software and the routine returns DESERR_NOHWDEVICE. For *cbc_crypt*, the parameter *ivec* is the the 8-byte initialization vector for the chaining. It is updated to the next initialization vector upon return.

SEE ALSO

**des**(1), **crypt**(3)

DIAGNOSTICS

| | |
|---|---|
| DESERR_NONE | No error. |
| DESERR_NOHWDEVICE | Encryption succeeded, but done in software instead of the requested hardware. |
| DESERR_HWERR | An error occurred in the hardware or driver. |
| DESERR_BADPARAM | Bad parameter to routine. |

Given a result status *stat*, the macro DES_FAILED(stat) is false only for the first two statuses.

RESTRICTIONS

These routines are not available for export outside the U.S.

NAME

directory, opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(filename)
char *filename;

struct dirent
*readdir(dirp)
DIR *dirp;

long
telldir(dirp)
DIR *dirp;

seekdir(dirp, loc)
DIR *dirp;
long loc;

rewinddir(dirp)
DIR *dirp;

closedir(dirp)
DIR *dirp;
```

DESCRIPTION

**opendir( )** opens the directory named by *filename* and associates a **directory stream** with it. **opendir( )** returns a pointer to be used to identify the **directory stream** in subsequent operations. The pointer NULL is returned if *filename* cannot be accessed or is not a directory, or if it cannot **malloc**(3) enough memory to hold the whole thing.

**readdir( )** returns a pointer to the next directory entry. It returns NULL upon reaching the end of the directory or detecting an invalid **seekdir( )** operation.

**telldir( )** returns the current location associated with the named **directory stream.**

**seekdir( )** sets the position of the next **readdir( )** operation on the *directory stream*. The new position reverts to the one associated with the **directory stream** when the **telldir( )** operation was performed. Values returned by **telldir( )** are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the **telldir( )** value may be invalidated due to undetected directory compaction. It is safe to use a previous **telldir( )** value immediately after a call to **opendir( )** and before any calls to **readdir**.

**rewinddir( )** resets the position of the named **directory stream** to the beginning of the directory.

**closedir( )** closes the named **directory stream** and frees the structure associated with the DIR pointer.

EXAMPLES

Sample code which searchs a directory for entry ''name'' is:

```
dirp = opendir(".");
for (dp = readdir(dirp); dp != NULL; dp = readdir(dirp))
        if (!strcmp(dp->d_name, name)) {
                closedir (dirp);
                return FOUND;
        }
closedir (dirp);
return NOT_FOUND;
```

NOTES

**The directory( )** library routines now use a new include file, **<dirent.h>**. This replaces the file, **<sys/dir.h>**, used in previous releases. Furthermore, with the use of this new file, the **readdir( )** routine returns directory entries whose structure is named **struct dirent** rather than **struct direct** as before. The file, **<sys/dir.h>**, is retained in the current SunOS release for purposes of backwards source code compatibility; programs which use the **directory( )** library and the file, **<sys/dir.h>**, will continue to compile and run without source code modifications. However, existing programs should convert to the use of the new include file, **<dirent.h>**, as **<sys/dir.h>** will be removed in a future major release.

SEE ALSO

**close**(2), **lseek**(2), **open**(2V), **read**(2V), **getwd**(3), **malloc**(3), **dir**(5)

NAME

drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 – generate uniformly distributed pseudo-random numbers

SYNOPSIS

**double drand48()**

**double erand48(xsubi)**
**unsigned short xsubi[3];**

**long lrand48()**

**long nrand48(xsubi)**
**unsigned short xsubi[3];**

**long mrand48()**

**long jrand48(xsubi)**
**unsigned short xsubi[3];**

**void srand48(seedval)**
**long seedval;**

**unsigned short *seed48(seed16v)**
**unsigned short seed16v[3];**

**void lcong48(param)**
**unsigned short param[7];**

DESCRIPTION

This family of functions generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic.

Functions **drand48()** and **drand48()** return non-negative double-precision floating-point values uniformly distributed over the interval (0.0, 1.0).

Functions **drand48()** and **drand48()** return non-negative long integers uniformly distributed over the interval $(0, 2^{31})$.

Functions **drand48()** and **drand48()** return signed long integers uniformly distributed over the interval $(-2^{31}, 2^{31})$.

Functions **drand48, seed48,** and **lcong48()** are initialization entry points, one of which should be invoked before either **drand48, drand48,** or **drand48()** is called. (Although it is not recommended practice, constant default initializer values will be supplied automatically if **drand48, drand48,** or **drand48()** is called without a prior call to an initialization entry point.) Functions **drand48, drand48,** and **drand48()** do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, $X_i$, according to the linear congruential formula

$$X_{n+1} = (aX_n + c)_{\text{mod } m} \qquad n \geq 0.$$

The parameter $m = 2^{48}$; hence 48-bit integer arithmetic is performed. Unless **lcong48()** has been invoked, the multiplier value $a$ and the addend value $c$ are given by

$$a = 5\text{DEECE66D}_{16} = 273673163155_8$$
$$c = \text{B}_{16} = 13_8.$$

The value returned by any of the functions **drand48, drand48, drand48, drand48, drand48,** or **drand48()** is computed by first generating the next 48-bit $X_i$ in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of $X_i$ and transformed into the returned value.

The functions **drand48**, **drand48**, and **drand48()** store the last 48-bit $X_i$ generated in an internal buffer; that is why they must be initialized prior to being invoked. The functions **drand48**, **drand48**, and **drand48()** require the calling program to provide storage for the successive $X_i$ values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of $X_i$ into the array and pass it as an argument. By using different arguments, functions **drand48**, **drand48**, and **drand48()** allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers, that is, the sequence of numbers in each stream will *not* depend upon how many times the routines have been called to generate numbers for the other streams.

The initializer function **drand48()** sets the high-order 32 bits of $X_i$ to the 32 bits contained in its argument. The low-order 16 bits of $X_i$ are set to the arbitrary value $330E_{16}$.

The initializer function **seed48()** sets the value of $X_i$ to the 48-bit value specified in the argument array. In addition, the previous value of $X_i$ is copied into a 48-bit internal buffer, used only by **seed48**, and a pointer to this buffer is the value returned by **seed48**. This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time — use the pointer to get at and store the last $X_i$ value, and then use this value to reinitialize via **seed48()** when the program is restarted.

The initialization function **lcong48()** allows the user to specify the initial $X_i$, the multiplier value $a$, and the addend value $c$. Argument array elements *param*[0-2] specify $X_i$, *param*[3-5] specify the multiplier $a$, and *param*[6] specifies the 16-bit addend $c$. After **lcong48()** has been called, a subsequent call to either **drand48()** or **seed48()** will restore the "standard" multiplier and addend values, $a$ and $c$, specified on the previous page.

SEE ALSO

**rand**(3C)

NAME

econvert, fconvert, gconvert, seconvert, sfconvert, sgconvert, ecvt, fcvt, gcvt – output conversion

SYNOPSIS

```
#include <floatingpoint.h>

char *econvert(value, ndigit, decpt, sign, buf)
double value;
int ndigit, *decpt, *sign;
char *buf;

char *fconvert(value, ndigit, decpt, sign, buf)
double value;
int ndigit, *decpt, *sign;
char *buf;

char *gconvert(value, ndigit, trailing, buf)
double value;
int ndigit;
int trailing;
char *buf;

char *seconvert(value, ndigit, decpt, sign, buf)
single *value;
int ndigit, *decpt, *sign;
char *buf;

char *sfconvert(value, ndigit, decpt, sign, buf)
single *value;
int ndigit, *decpt, *sign;
char *buf;

char *sgconvert(value, ndigit, trailing, buf)
single *value;
int ndigit;
int trailing;
char *buf;

char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
int ndigit;
char *buf;
```

DESCRIPTION

**econvert()** converts the *value* to a NULL-terminated string of *ndigit* ASCII digits in *buf* and returns a pointer to *buf*. *buf* should contain at least *ndigit+1* characters. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt*. Thus *buf* == "314" and **decpt* == 1 corresponds to the numerical value 3.14, while *buf* == "314" and **decpt* == −1 corresponds to the numerical value .0314. If the sign of the result is negative, the word pointed to by *sign* is nonzero; otherwise it is zero. The least significant digit is rounded.

**fconvert** is identical to **econvert**, except that the correct digit has been rounded for Fortran F-format output with *ndigit* digits to the right of the decimal point. *ndigit* can be negative to indicate rounding to the left of the decimal point. The return value is a pointer to *buf*. *buf* should contain at least *310+max(0,ndigit)* characters to accomodate any double-precision *value*.

**gconvert( )** converts the *value* to a NULL-terminated ASCII string in *buf* and returns a pointer to *buf*. It produces *ndigit* significant digits in fixed-decimal format, like Fortran F, if possible, and otherwise in floating-decimal format, like Fortran E; in either case *buf* is ready for printing, with sign and exponent. The result corresponds to that obtained by

**(void) sprintf(buf,"%gw.n",value) ;**

If *trailing*= 0, trailing zeros and a trailing point are suppressed. If *trailing*!= 0, trailing zeros and a trailing point are retained.

**seconvert, sfconvert,** and **sgconvert( )** are single-precision versions of these functions, and are more efficient than the corresponding double-precision versions. A pointer rather than the value itself is passed to avoid C's usual conversion of single-precision arguments to double.

**ecvt( )** and **fcvt( )** are obsolete versions of **econvert( )** and **fconvert( )** that create a string in a static data area, overwritten by each call, and return values that point to that static data. These functions are therefore not reentrant.

**gcvt( )** is an obsolete version of **gconvert( )** that always suppresses trailing zeros and point.

IEEE Infinities and NaNs are treated similarly by these functions. ''NaN'' is returned for NaN, and ''Inf'' or ''Infinity'' for Infinity. The longer form is produced when *ndigit* >= 8.

SEE ALSO

**printf**(3S)

## NAME

end, etext, edata – last locations in program

## SYNOPSIS

```
extern end;
extern etext;
extern edata;
```

## DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and **end()** above the uninitialized data region.

When execution begins, the program break (the first location beyond the data) coincides with **end**, but it is reset by the routines **brk**(2), **malloc**(3), standard input/output (**stdio**(3S) and **stdio**(3V)), the profile (**–p**) option of **cc**(1V), and so on. Thus, the current value of the program break should be determined by **sbrk(0)** (see **brk**(2)).

## SEE ALSO

**cc**(1V), **brk**(2), **malloc**(3), **stdio**(3S), **stdio**(3V)

NAME

ethers, ether_ntoa, ether_aton, ether_ntohost, ether_hostton, ether_line – Ethernet address mapping operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>

char *
ether_ntoa(e)
        struct ether_addr *e;

struct ether_addr *
ether_aton(s)
        char *s;

ether_ntohost(hostname, e)
        char *hostname;
        struct ether_addr *e;

ether_hostton(hostname, e)
        char *hostname;
        struct ether_addr *e;

ether_line(l, e, hostname)
        char *l;
        struct ether_addr *e;
        char *hostname;
```

DESCRIPTION

These routines are useful for mapping 48 bit Ethernet numbers to their ASCII representations or their corresponding host names, and vice versa.

The function **ether_ntoa( )** converts a 48 bit Ethernet number pointed to by *e* to its standard ACSII representation; it returns a pointer to the ASCII string. The representation is of the form: *x:x:x:x:x:x* where *x* is a hexadecimal number between 0 and ff. The function **ether_aton( )** converts an ASCII string in the standard representation back to a 48 bit Ethernet number; the function returns NULL if the string cannot be scanned successfully.

The function **ether_ntohost( )** maps an Ethernet number (pointed to by *e*) to its associated hostname. The string pointed to by **hostname** must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. Inversely, the function **ether_hostton( )** maps a hostname string to its corresponding Ethernet number; the function modifies the Ethernet number pointed to by *e*. The function also returns zero upon success and non-zero upon failure.

The function **ether_line( )** scans a line (pointed to by *l*) and sets the hostname and the Ethernet number (pointed to by *e*). The string pointed to by **hostname** must be long enough to hold the hostname and a NULL character. The function returns zero upon success and non-zero upon failure. The format of the scanned line is described by **ethers**(5).

FILES

**/etc/ethers** (or the Yellow Pages maps **ethers.byaddr** and **ethers.byname**)

SEE ALSO

**ethers**(5)

## NAME

execl, execv, execle, execlp, execvp – execute a file

## SYNOPSIS

```
execl(name, arg0, arg1, ..., argn, (char *)0)
char *name, *arg0, *arg1, ..., *argn;

execv(name, argv)
char *name, *argv[ ];

execle(name, arg0, arg1, ..., argn, (char *)0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[ ];

execlp(name, arg0, arg1, ..., argn, (char *)0)
char *name, *arg0, *arg1, ..., *argn;

execvp(name, argv)
char *name, *argv[ ];

extern char **environ;
```

## DESCRIPTION

These routines provide various interfaces to the **execve()** system call. Refer to **execve**(2) for a description of their properties; only brief descriptions are provided here.

**exec** in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

The *filename* argument is a pointer to the name of the file to be executed. The pointers *arg*[0], *arg*[1]... address NULL-terminated strings. Conventionally *arg*[0] is the name of the file.

Two interfaces are available. **execl()** is useful when a known file with known arguments is being called; the arguments to **execl()** are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component). A **(char *)0** argument must end the argument list. The cast to type **char *** insures portability.

The **execv()** version is useful when the number of arguments is unknown in advance; the arguments to **execv()** are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file.

*argv* is directly usable in another **execv()** because *argv*[*argc*] is 0.

*envp* is a pointer to an array of strings that constitute the *environment* of the process. Each string consists of a name, an '=', and a NULL-terminated value. The array of pointers is terminated by a NULL pointer. The shell **sh**(1) passes an environment entry for each global shell variable defined when the program is called. See **environ**(5V) for some conventionally used names. The C run-time start-off routine places a copy of *envp* in the global cell *environ*, which is used by **execv()** and **execl()** to pass the environment to any subprograms executed by the current program.

**execlp()** and **execvp()** are called with the same arguments as **execl()** and **execv**, but duplicate the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

FILES

| | |
|---|---|
| **/usr/bin/sh** | shell, invoked if command file found by **execlp( )** or **execvp( )** |

SEE ALSO

**csh**(1), **sh**(1), **execve**(2), **fork**(2), **a.out**(5), **environ**(5V)

*Programming Utilities and Libraries*

DIAGNOSTICS

If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see **a.out**(5)), if maximum memory is exceeded, or if the arguments require too much space, a return constitutes the diagnostic; the return value is −1. Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

NAME

exit – terminate a process after performing cleanup

SYNOPSIS

**exit(status)**
**int status;**

DESCRIPTION

**exit()** terminates a process by calling **exit**(2) after calling any termination handlers named by calls to **on_exit**. Normally, this is just the Standard I/O library function **_cleanup**. **exit()** never returns.

SEE ALSO

**exit**(2), **intro**(3S), **on_exit**(3)

## NAME

exportent, getexportent, setexportent, addexportent, remexportent, endexportent, getexportopt – get exported file system information

## SYNOPSIS

```
#include <stdio.h>
#include <exportent.h>
FILE *setexportent( )
struct exportent *getexportent(filep)
        FILE *filep;
int addexportent(filep, dirname, options)
        FILE *filep;
        char *dirname;
        char *options;
int remexportent(filep, dirname)
        FILE *filep;
        char *dirname;
char *getexportopt(xent, opt)
        struct exportent *xent;
        char *opt;
void endexportent(filep)
        FILE *filep;
```

## DESCRIPTION

These routines access the exported filesystem information in **/etc/xtab**.

**setexportent( )** opens the export information file and returns a file pointer to use with **getexportent**, **addexportent**, **remexportent**, and **endexportent**. **getexportent( )** reads the next line from *filep* and returns a pointer to an object with the following structure containing the broken-out fields of a line in the file, **/etc/xtab** The fields have meanings described in **exports**(5).

```
#define ACCESS_OPT  "access"  /* machines that can mount fs */
#define ROOT_OPT    "root"    /* machines with root access of fs */
#define RO_OPT      "ro"      /* export read-only */
#define ANON_OPT    "anon"    /* uid for anonymous requests */
#define SECURE_OPT  "secure"  /* require secure NFS for access */
#define WINDOW_OPT  "window"  /* expiration window for credential */
struct exportent {
        char *xent_dirname;     /* directory (or file) to export */
        char *xent_options;     /* options, as above */
};
```

**addexportent( )** adds the **exportent( )** to the end of the open file *filep*. It returns 0 if successful and −1 on failure. **remexportent( )** removes the indicated entry from the list. It also returns 0 on success and −1 on failure. **getexportopt( )** scans the *xent_options* field of the **exportent( )** structure for a substring that matches *opt*. It returns the string value of *opt*, or NULL if the option is not found.

**endexportent( )** closes the file.

## FILES

**/etc/exports**
**/etc/xtab**

## SEE ALSO

**exports**(5), **exportfs**(8)

**DIAGNOSTICS**

NULL pointer (0) returned on EOF or error.

**BUGS**

The returned **exportent( )** structure points to static information that is overwritten in each call.

NAME

fclose, fflush – close or flush a stream

SYNOPSIS

**#include <stdio.h>**

**fclose(stream)**
**FILE *stream;**

**fflush(stream)**
**FILE *stream;**

DESCRIPTION

**fclose( )** writes out any buffered data for the named stream, and closes the named stream. Buffers allocated by the standard input/output system are freed.

**fclose( )** is performed automatically for all open files upon calling **exit**(3).

**fflush( )** writes out any buffered data for the named output stream. The named stream remains open.

SEE ALSO

**close**(2), **exit**(3), **fopen**(3S), **setbuf**(3S)

DIAGNOSTICS

These functions return 0 for success, and EOF if any error (such as trying to write to a file that has not been opened for writing) was detected.

NAME

fdate – return date and time in an ASCII string

SYNOPSIS

**subroutine fdate(string)**
**character*24 string**

**character*24 function fdate( )**

DESCRIPTION

**fdate( )** returns the current date and time as a 24 character string in the format described under **ctime**(3). Neither NEWLINE nor NULL will be included.

**fdate( )** can be called either as a function or as a subroutine. If called as a function, the calling routine must define its type and length. For example:

```
character*24 fdate
write(*,*)fdate( )
```

SEE ALSO

**ctime**(3), **time**(3F)

NAME

ferror, feof, clearerr, fileno – stream status inquiries

SYNOPSIS

```
#include <stdio.h>

ferror(stream)
FILE *stream;

feof(stream)
FILE *stream;

clearerr(stream)
FILE *stream;

fileno(stream)
FILE *stream;
```

DESCRIPTION

**ferror( )** returns non-zero when an error has occurred reading from or writing to the named stream, otherwise zero. Unless cleared by **clearerr**, the error indication lasts until the stream is closed.

**feof( )** returns non-zero when EOF has previously been detected reading the named input stream, otherwise zero. Unless cleared by **clearerr**, the EOF indication lasts until the stream is closed.

**clearerr( )** resets the error indication and EOF indication to zero on the named stream.

**fileno( )** returns the integer file descriptor associated with the stream; see **open**(2V).

NOTE

All these functions are implemented as macros; they cannot be redeclared.

SEE ALSO

**open**(2V), **fopen**(3S)

## NAME

single_to_decimal, double_to_decimal, extended_to_decimal – convert floating-point value to decimal record

## SYNOPSIS

```
#include <floatingpoint.h>

void single_to_decimal(px, pm, pd, ps)
single *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void double_to_decimal(px, pm, pd, ps)
double *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;

void extended_to_decimal(px, pm, pd, ps)
extended *px ;
decimal_mode *pm;
decimal_record *pd;
fp_exception_field_type *ps;
```

## DESCRIPTION

The **floating_to_decimal()** functions convert the floating-point value at **px* into a decimal record at **pd*, observing the modes specified in **pm* and setting exceptions in **ps*. If there are no IEEE exceptions, **ps* will be zero.

If **px* is zero, infinity, or NaN, then only *pd->sign* and *pd->fpclass* are set. Otherwise *pd->exponent* and *pd->ds* are also set so that

**(pd->sign)*(pd->ds)*10**(pd->exponent)**

is a correctly rounded approximation to **px*. *pd->ds* has at least one and no more than DECIMAL_STRING_LENGTH–1 significant digits because one character is used to terminate the string with a NULL.

*pd->ds* is correctly rounded according to the IEEE rounding modes in *pm->rd*. **ps* has *fp_inexact* set if the result was inexact, and has *fp_overflow* set if the string result does not fit in *pd->ds* because of the limitation DECIMAL_STRING_LENGTH.

If *pm->df == floating_form*, then *pd->ds* always contains *pm->ndigits* significant digits. Thus if **px* == 12.34 and *pm->ndigits* == 8, then *pd->ds* will contain 12340000 and *pd->exponent* will contain –6.

If *pm->df == fixed_form* and *pm->ndigits* >= 0, then *pd->ds* always contains *pm->ndigits* after the point and as many digits as necessary before the point. Since the latter is not known in advance, the total number of digits required is returned in *pd->ndigits*; if that number >= DECIMAL_STRING_LENGTH, then *ds* is undefined. *pd->exponent* always gets *–pm->ndigits*. Thus if **px* == 12.34 and *pm->ndigits* == 1, then *pd->ds* gets 123, *pd->exponent* gets –1, and *pd->ndigits* gets 3.

If *pm->df == fixed_form* and *pm->ndigits* < 0, then *pm->ds* always contains *–pm->ndigits* trailing zeros; in other words, rounding occurs *–pm->ndigits* to the left of the decimal point, but the digits rounded away are retained as zeros. The total number of digits required is in *pd->ndigits*. *pd->exponent* always gets 0. Thus if **px* == 12.34 and *pm->ndigits* == –1, then *pd->ds* gets 10, *pd->exponent* gets 0, and *pd->ndigits* gets 2.

*pd->more* is not used.

**econvert**(3), **fconvert**, **gconvert**, **printf**(3S), and **sprintf**, all use **double_to_decimal**.

SEE ALSO

**econvert**(3), **printf**(3S)

NAME

floatingpoint – IEEE floating point definitions

SYNOPSIS

**#include <sys/ieeefp.h>**
**#include <floatingpoint.h>**

DESCRIPTION

This file defines constants, types, variables, and functions used to implement standard floating point according to ANSI/IEEE Std 754-1985. The variables and functions are implemented in **libc.a**. The included file **<sys/ieeefp.h>** defines certain types of interest to the kernel.

IEEE Rounding Modes:

**fp_direction_type** The type of the IEEE rounding direction mode. Note: the order of enumeration varies according to hardware.

**fp_direction** The IEEE rounding direction mode currently in force. This is a global variable that is intended to reflect the hardware state, so it should only be written indirectly through a function like " .}S 3 2 "**ieee_flags(set,direction,...)**"" "" "" "" "" "" that also sets the hardware state.

**fp_precision_type** The type of the IEEE rounding precision mode, which only applies on systems that support extended precision such as Sun-3 systems with 68881's.

**fp_precision** The IEEE rounding precision mode currently in force. This is a global variable that is intended to reflect the hardware state on systems with extended precision, so it should only be written indirectly through a function like **ieee_flags("set","precision",...)**.

SIGFPE handling:

**sigfpe_code_type** The type of a SIGFPE code.

**sigfpe_handler_type** The type of a user-definable SIGFPE exception handler called to handle a particular SIGFPE code.

SIGFPE_DEFAULT A macro indicating the default SIGFPE exception handling, namely to perform the exception handling specified by calls to **ieee_handler(3M)**, if any, and otherwise to dump core using **abort**(3).

SIGFPE_IGNORE A macro indicating an alternate SIGFPE exception handling, namely to ignore and continue execution.

SIGFPE_ABORT A macro indicating an alternate SIGFPE exception handling, namely to abort with a core dump.

IEEE Exception Handling:

N_IEEE_EXCEPTION The number of distinct IEEE floating-point exceptions.

**fp_exception_type** The type of the N_IEEE_EXCEPTION exceptions. Each exception is given a bit number.

**fp_exception_field_type**
The type intended to hold at least N_IEEE_EXCEPTION bits corresponding to the IEEE exceptions numbered by **fp_exception_type**. Thus **fp_inexact** corresponds to the least significant bit and **fp_invalid** to the fifth least significant bit. Note: some operations may set more than one exception.

**fp_accrued_exceptions**
The IEEE exceptions between the time this global variable was last cleared, and the last time a function like **ieee_flags("get","exception",...)** was called to update the variable by obtaining the hardware state.

**ieee_handlers** An array of user-specifiable signal handlers for use by the standard **SIGFPE** handler for IEEE arithmetic-related **SIGFPE** codes. Since IEEE trapping modes correspond to hardware modes, elements of this array should only be modified with a function like **ieee_handler**(3M) that performs the appropriate hardware mode update. If no **sigfpe_handler** has been declared for a particular IEEE-related **SIGFPE** code, then the related **ieee_handlers** will be invoked.

IEEE Formats and Classification:

*single*;*extended* Definitions of IEEE formats.

**fp_class_type** An enumeration of the various classes of IEEE values and symbols.

IEEE Base Conversion:

The functions described under **floating_to_decimal**(3) and **decimal_to_floating**(3) not only satisfy the IEEE Standard, but also the stricter requirements of correct rounding for all arguments.

DECIMAL_STRING_LENGTH
The length of a **decimal_string**.

**decimal_string** The digit buffer in a **decimal_record**.

**decimal_record** The canonical form for representing an unpacked decimal floating-point number.

**decimal_form** The type used to specify fixed or floating binary to decimal conversion.

**decimal_mode** A struct that contains specifications for conversion between binary and decimal.

**decimal_string_form** An enumeration of possible valid character strings representing floating-point numbers, infinities, or NaNs.

FILES

**/usr/include/sys/ieeefp.h**
**/usr/include/floatingpoint.h**
**/usr/lib/libc.a**

SEE ALSO

**abort**(3), **decimal_to_floating**(3), **econvert**(3), **floating_to_decimal**(3), **ieee_flags**(3M), **ieee_handler**(3M), **sigfpe**(3), **string_to_decimal**(3), **strtod**(3)

NAME

fopen, freopen, fdopen – open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

DESCRIPTION

**fopen()** opens the file named by *filename* and associates a stream with it. If the open succeeds, **fopen()** returns a pointer to be used to identify the stream in subsequent operations.

*filename* points to a character string that contains the name of the file to be opened.

*type* is a character string having one of the following values:

| | |
|---|---|
| **r** | open for reading |
| **w** | truncate or create for writing |
| **a** | append: open for writing at end of file, or create for writing |
| **r+** | open for update (reading and writing) |
| **w+** | truncate or create for update |
| **a+** | append; open or create for update at EOF |

**freopen()** opens the file named by *filename* and associates the stream pointed to by *stream* with it. The *type* argument is used just as in **fopen**. The original stream is closed, regardless of whether the open ultimately succeeds. If the open succeeds, **freopen()** returns the original value of *stream*.

**freopen()** is typically used to attach the preopened streams associated with **stdin, stdout,** and **stderr** to other files.

**fdopen()** associates a stream with the file descriptor *fildes*. File descriptors are obtained from calls like **open, dup, creat,** or **pipe**(2), which open files but do not return streams. Streams are necessary input for many of the Section 3S library routines. The *type* of the stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening **fseek()** or **rewind,** and input may not be directly followed by output without an intervening **fseek, rewind,** or an input operation which encounters end-of-file.

SEE ALSO

**open**(2V), **pipe**(2), **fclose**(3S), **fopen**(3V), **fseek**(3S)

DIAGNOSTICS

**fopen, freopen,** and **fdopen()** return a NULL pointer on failure.

BUGS

In order to support the same number of open files that the system does, **fopen()** must allocate additional memory for data structures using **calloc()** after 64 files have been opened. This confuses some programs which use their own memory allocators.

NAME

fread, fwrite – buffered binary input/output

SYNOPSIS

**#include <stdio.h>**

**fread(ptr, size, nitems, stream)**
**FILE *stream;**

**fwrite(ptr, size, nitems, stream)**
**FILE *stream;**

DESCRIPTION

**fread( )** reads, into a block pointed to by *ptr*, *nitems* of data from the named input stream, where an item of data is a sequence of bytes (not necessarily terminated by a NULL byte) of length *size*. It returns the number of items actually read. **fread( )** stops appending bytes if an EOF or error condition is encountered while reading stream, or if *nitems* items have been read. **fread( )** leaves the file pointer in stream, if defined, pointing to the byte following the last byte read if there is one. **fread( )** does not change the contents of stream.

**fwrite( )** appends at most *nitems* of data from the block pointed to by *ptr* to the named output stream. It returns the number of items actually written. **fwrite( )** stops appending when it has appended *nitems* items of data or if an error condition is encountered on stream. **fwrite( )** does not change the contents of the block pointed to by *ptr*.

The argument **size** is typically **sizeof(*ptr)** where the pseudo-function **sizeof** specifies the length of an item pointed to by *ptr*. If *ptr* points to a data type other than *char* it should be cast into a pointer to *char*.

If *size* or *nitems* is non-positive, no characters are read or written and 0 is returned by both **fread( )** and **fwrite( )**.

SEE ALSO

**read**(2V), **write**(2V), **fopen**(3S), **getc**(3S), **gets**(3S), **putc**(3S), **puts**(3S), **printf**(3S), **scanf**(3S),

DIAGNOSTICS

**fread( )** and **fwrite( )** return 0 upon end of file or error.

NAME

fseek, ftell, rewind – reposition a stream

SYNOPSIS

**#include <stdio.h>**

**fseek(stream, offset, ptrname)**
**FILE *stream;**
**long offset;**

**long ftell(stream)**
**FILE *stream;**

**rewind(stream)**
**FILE *stream;**

DESCRIPTION

**fseek( )** sets the position of the next input or output operation on the stream. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

**rewind(***stream***)** is equivalent to **fseek(***stream*, 0L, 0), except that no value is returned.

**fseek( )** and **rewind( )** undo any effects of **ungetc**(3S).

After **fseek( )** or **rewind**, the next operation on a file opened for update may be either input or output.

**ftell( )** returns the offset of the current byte relative to the beginning of the file associated with the named stream.

SEE ALSO

**lseek**(2), **fopen**(3S), **popen**(3S), **ungetc**(3S)

DIAGNOSTICS

**fseek( )** returns –1 for improper seeks, otherwise zero. An improper seek can be, for example, an **fseek( )** done on a file that has not been opened using **fopen**; in particular, **fseek( )** may not be used on a terminal, or on a file opened using **popen**(3S).

WARNING

Although on the UNIX system an offset returned by **ftell( )** is measured in bytes, and it is permissible to seek to positions relative to that offset, portability to a (non-UNIX) system requires that an offset be used by **fseek( )** directly. Arithmetic may not meaningfully be performed on such an offset, which is not necessarily measured in bytes.

NAME

ftok – standard interprocess communication package

SYNOPSIS

**#include <sys/types.h>**
**#include <sys/ipc.h>**

**key_t ftok(path, id)**
**char *path;**
**char id;**

DESCRIPTION

All interprocess communication facilities require the user to supply a key to be used by the **msgget**(2), **semget**(2), and **shmget**(2) system calls to obtain interprocess communication identifiers. One suggested method for forming a key is to use the **ftok( )** subroutine described below. Another way to compose keys is to include the project ID in the most significant byte and to use the remaining portion as a sequence number. There are many other ways to form keys, but it is necessary for each system to define standards for forming them. If some standard is not adhered to, it will be possible for unrelated processes to unintentionally interfere with each other's operation. Therefore, it is strongly suggested that the most significant byte of a key in some sense refer to a project so that keys do not conflict across a given system.

**ftok( )** returns a key based on *path* and ID that is usable in subsequent **msgget, semget,** and **shmget( )** system calls. *path* must be the path name of an existing file that is accessible to the process. ID is a character which uniquely identifies a project. Note: **ftok( )** will return the same key for linked files when called with the same ID and that it will return different keys when called with the same file name but different IDs.

SEE ALSO

**intro**(2), **msgget**(2), **semget**(2), **shmget**(2)

DIAGNOSTICS

**ftok( )** returns **(key_t) –1** if *path* does not exist or if it is not accessible to the process.

WARNING

If the file whose *path* is passed to **ftok( )** is removed when keys still refer to the file, future calls to **ftok( )** with the same *path* and ID will return an error. If the same file is recreated, then **ftok( )** is likely to return a different key than it did the original time it was called.

NAME

ftw – walk a file tree

SYNOPSIS

**#include <ftw.h>**

**int ftw(path, fn, depth)**
**char *path;**
**int (*fn)();**
**int depth;**

DESCRIPTION

**ftw()** recursively descends the directory hierarchy rooted in *path*. For each object in the hierarchy, **ftw()** calls *fn*, passing it a pointer to a NULL-terminated character string containing the name of the object, a pointer to a **stat()** structure (see **stat**(2)) containing information about the object, and an integer. Possible values of the integer, defined in the **<ftw.h>** header file, are FTW_F for a file, FTW_D for a directory, FTW_DNR for a directory that cannot be read, and FTW_NS for an object for which **stat()** could not successfully be executed. If the integer is FTW_DNR, descendants of that directory will not be processed. If the integer is FTW_NS, the **stat()** structure will contain garbage. An example of an object that would cause FTW_NS to be passed to *fn* would be a file in a directory with read but without execute (search) permission.

**ftw()** visits a directory before visiting any of its descendants.

The tree traversal continues until the tree is exhausted, an invocation of *fn* returns a nonzero value, or some error is detected within **ftw()** (such as an I/O error). If the tree is exhausted, **ftw()** returns zero. If *fn* returns a nonzero value, **ftw()** stops its tree traversal and returns whatever value was returned by *fn*. If **ftw()** detects an error, it returns –1, and sets the error type in **errno**.

**ftw()** uses one file descriptor for each level in the tree. The *depth* argument limits the number of file descriptors so used. If *depth* is zero or negative, the effect is the same as if it were 1. *depth* must not be greater than the number of file descriptors currently available for use. **ftw()** will run more quickly if *depth* is at least as large as the number of levels in the tree.

SEE ALSO

**stat**(2), **malloc**(3)

BUGS

Because **ftw()** is recursive, it is possible for it to terminate with a memory fault when applied to very deep file structures.

It could be made to run faster and use less storage on deep structures at the cost of considerable complexity.

**ftw()** uses **malloc**(3) to allocate dynamic storage during its operation. If **ftw()** is forcibly terminated, such as by **longjmp()** being executed by *fn* or an interrupt routine, **ftw()** will not have a chance to free that storage, so it will remain permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have *fn* return a nonzero value at its next invocation.

NAME

getacinfo, getacdir, getacflg, getacmin, setac, endac – get audit control file information

SYNOPSIS

**int getacdir(dir, len)**
**char *dir;**
**int len;**

**int getacmin(min_val)**
**int *min_val;**

**int getacflg(auditstring, len)**
**char *auditstring;**
**int len;**

**void setac( )**

**void endac( )**

DESCRIPTION

When first called, **getacdir( )** provides information about the first audit directory in the **audit_control** file; thereafter, it returns the next directory in the file. Successive calls list all the directories listed in **audit_control**(5) The parameter *len* specifies the length of the buffer *dir* . On return, *dir* points to the directory entry.

**getacmin( )** reads the minimum value from the **audit_control** file and returns the value in **min_val**. The minimum value specifies how full the file system to which the audit files are being written can get before the script **audit_warn** is invoked.

**getacflg( )** reads the system audit value from the **audit_control** file and returns the value in **auditstring**. The parameter *len* specifies the length of the buffer **auditstring**.

Calling *setac* rewinds the **audit_control** file to allow repeated searches.

Calling *endac* closes the **audit_control** file when processing is complete.

RETURN VALUE

Upon successful completion of all **getac...** functions, a zero is returned. **getacmin( )** and **getacflg( )** return a **1** on EOF. Only **getacdir( )** returns a **2** if the directory search had to start from the beginning because another **getac...** function was called between calls to **getacdir**.

ERRORS

Upon unsuccessful completion, a value of –2 is returned and **errno** is set to indicate the error. –3 is returned if the directory entry format in the. **audit_control** file is incorrect. If the input buffer is too short to accommodate the record, **getacdir( )** and **getacflg( )** return an error code of –3. Only **getacdir( )** returns a –1 on end of file.

SEE ALSO

**audit_control**(5)

NAME

getauditflagsbin, getauditflagschar – convert audit flag specifications

SYNOPSIS

**#include <sys/label.h>**
**#include <sys/audit.h>**
**#include <sys/auevents.h>**

**int getauditflagsbin(auditstring, masks)**
**char *auditstring;**
**audit_state_t *masks;**

**int getauditflagschar(auditstring, masks, verbose)**
**char *auditstring;**
**audit_state_t *masks;**
**int verbose;**

DESCRIPTION

**getauditflagsbin( )** converts the character representation of audit values pointed to by *auditstring* into **audit_state_t** fields pointed to by *masks*. These fields indicate which events are to be audited when they succeed and which are to be audited when they fail. The character string syntax is described in **audit_control**(5).

**getauditflagschar( )** converts the **audit_state_t** fields pointed to by *masks* into a string pointed to by *auditstring*. If *verbose* is zero, the short (2-character) flag names are used. If *verbose* is non-zero, the long flag names are used. *auditstring* should be large enough to contain the ASCII representation of the events.

*auditstring* contains a series of event names, each one identifying a single audit class, separated by commas. The **audit_state_t** fields pointed to by *masks* correspond to binary values defined in *audit.h*.

DIAGNOSTICS

–1 is returned on error and 0 on success.

SEE ALSO

**audit.log**(5), **audit_control**(5)

BUGS

This is not a very extensible interface.

NAME

getc, getchar, fgetc, getw – get character or integer from stream

SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;
```

DESCRIPTION

**getc**( ) returns the next character (that is, byte) from the named input stream, as an integer. It also moves the file pointer, if defined, ahead one character in stream. **getchar**( ) is defined as **getc(stdin)**. **getc** and **getchar** are macros.

**fgetc**( ) behaves like **getc**, but is a function rather than a macro. **fgetc**( ) runs more slowly than **getc**, but it takes less space per invocation and its name can be passed as an argument to a function.

**getw**( ) returns the next C **int** (*word*) from the named input stream. **getw**( ) increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. **getw**( ) assumes no special alignment in the file.

SEE ALSO

**ferror**(3S), **fopen**(3S), **fread**(3S), **gets**(3S), **putc**(3S), **scanf**(3S), **ungetc**(3S)

DIAGNOSTICS

These functions return the integer constant EOF at EOF or upon an error. The EOF condition is remembered, even on a terminal, and all subsequent attempts to read will return EOF until the condition is cleared with **clearerr** (see **ferror**(3S)) Because EOF is a valid integer, **ferror**(3S) should be used to detect **getw**( ) errors.

WARNING

If the integer value returned by **getc**, **getchar**, or **fgetc** is stored into a character variable and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a character on widening to integer is machine-dependent.

BUGS

Because it is implemented as a macro, **getc**( ) treats a stream argument with side effects incorrectly. In particular, **getc(*f++)** does not work sensibly. **fgetc**( ) should be used instead.

Because of possible differences in word length and byte ordering, files written using **putw**( ) are machine-dependent, and may not be readable using **getw**( ) on a different processor.

NAME

getcwd – get pathname of current working directory

SYNOPSIS

```
char *getcwd (buf, size)
char *buf;
int size;
```

DESCRIPTION

**getcwd()** returns a pointer to the current directory pathname. The value of *size* must be at least two greater than the length of the pathname to be returned.

If *buf* is a NULL pointer, **getcwd()** will obtain *size* bytes of space using **malloc**(3). In this case, the pointer returned by **getcwd()** may be used as the argument in a subsequent call to **free**.

The function is implemented by using **popen**(3S) to pipe the output of the **pwd**(1) command into the specified string space.

EXAMPLE

```
char *cwd, *getcwd();
.
.
.
if ((cwd = getcwd((char *)NULL, 64)) == NULL) {
        perror ("pwd");
        exit (1);
}
printf("%s\n", cwd);
```

SEE ALSO

**malloc**(3), **popen**(3S), **pwd**(1)

DIAGNOSTICS

Returns NULL with **errno** set if *size* is not large enough, or if an error ocurrs in a lower-level function.

BUGS

Since this function uses **popen()** to create a pipe to the **pwd** command, it is slower than **getwd()** and gives poorer error diagnostics. **getcwd()** is provided only for compatibility with other UNIX operating systems.

NAME

getenv – return value for environment name

SYNOPSIS

**char *getenv(name)**
**char *name;**

DESCRIPTION

**getenv( )** searches the environment list (see **environ**(5V)) for a string of the form *name=value*, and returns a pointer to the string *value* if such a string is present, otherwise NULL pointer.

SEE ALSO

**environ**(5V), **execve**(2), **putenv**(3)

NAME

getfauditflags – generates the process audit state

SYNOPSIS

**#include <sys/types.h>**
**#include <sys/audit.h>**
**#include <sys/label.h>**

**void getfauditflags(usremasks, usrdmasks, lastmasks)**
**audit_state_t *usremasks;**
**audit_state_t *usrdmasks;**
**audit_state_t *lastmasks;**

DESCRIPTION

**getfauditflags** generates the process audit state from the user audit value as input to **getfauditflags** and the system audit value as specified in the **audit_control** file. **getfauditflags** obtains the system audit value by calling **getacflg**. The user audit value, pointed to by *usremasks* and *usrdmasks* is passed into **getfauditflags.**

*usremasks* points to audit_state_t fields which contains two values. The first value defines which events are *always* to be audited when they they **succeed.** The second value defines defines which events are *always* to be audited when they they **fail.**

*usrdmasks* also points to audit_state_t fields which contains two values. The first value defines which events are **never** to be audited when they they **succeed.** The second value defines defines which events are **never** to be audited when they they **fail.**

The structures pointed to by *usremasks* and *usrdmasks* may be obtained from the **passwd.adjunct** file by calling **getpwaent( )** which returns a pointer to a strucure containing all **passwd.adjunct** fields for a user.

*lastmasks* points to the return audit_state_t structure. This structure contains two values which define the process audit state. The first value defines which events are to be audited when they **succeed** and the second value defines which events are to be audited when they **fail.**

Both *usremasks* and *usrdmasks* override the values in the system audit values.

SEE ALSO

**getauditflags(3), getacinfo(3), audit.log(5), audit_control(5)**

## NAME

getfsent, getfsspec, getfsfile, getfstype, setfsent, endfsent – get file system descriptor file entry

## SYNOPSIS

```
#include <fstab.h>

struct fstab *getfsent( )

struct fstab *getfsspec(spec)
char *spec;

struct fstab *getfsfile(file)
char *file;

struct fstab *getfstype(type)
char *type;

int setfsent( )

int endfsent( )
```

## DESCRIPTION

These routines are included for compatibility with 4.2 BSD; they have been superseded by the **getmntent**(3) library routines.

**getfsent**, *getfsspec*, *getfstype*, and *getfsfile* each return a pointer to an object with the following structure containing the broken-out fields of a line in the file system description file, **<fstab.h>**.

```
struct fstab {
        char    *fs_spec;
        char    *fs_file;
        char    *fs_type;
        int     fs_freq;
        int     fs_passno;
};
```

The fields have meanings described in **fstab**(5).

**getfsent**( ) reads the next line of the file, opening the file if necessary.

**getfsent**( ) opens and rewinds the file.

*endfsent* closes the file.

*getfsspec* and *getfsfile* sequentially search from the beginning of the file until a matching special file name or file system file name is found, or until EOF is encountered. *getfstype* does likewise, matching on the file system type field.

## FILES

**/etc/fstab**

## SEE ALSO

**fstab**(5)

## DIAGNOSTICS

Null pointer (0) returned on EOF or error.

## BUGS

The return value points to static information which is overwritten in each call.

## NAME

getgraent, getgranam, setgraent, endgraent, fgetgraent – get group adjunct file entry

## SYNOPSIS

```
#include <grpadj.h>

struct group_adjunct *getgraent()

struct group_adjunct *getgranam(name)
char *name;

struct group_adjunct *fgetgraent(f)
FILE *f;

void setgraent()

void endgraent()
```

## DESCRIPTION

**getgraent()** and **getgranam()** each return pointers to an object with the following structure containing the broken-out fields of a line in the group adjunct file. Each line contains a **group_adjunct** structure, defined in the **<grpadj.h>** header file.

```
struct  group_adjunct {
char    *gra_name;      /* the name of the group */
char    *gra_passwd;    /* the encrypted group password */
};
```

When first called, **getgraent()** returns a pointer to a **group_adjunct** structure corresponding to the first line in the file. Thereafter, it returns a pointer to the next **group_adjunct** structure in the file. So successive calls may be used to traverse the entire file.

For locating a particular group, **getgranam()** searches through the file until it finds group *filename*, then returns a pointer to that structure.

A call to **getgraent()** rewinds the group adjunct file to allow repeated searches. A call to **endgraent()** closes the group adjunct file when processing is complete.

## SEE ALSO

**getlogin(3), getgrent(3), getpwaent(3), getpwent(3), ypserv(8)**

## DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

## BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent – get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent()

struct group *getgrgid(gid)
int gid;

struct group *getgrnam(name)
char *name;

setgrent()

endgrent()

struct group *fgetgrent(f)
FILE *f;
```

DESCRIPTION

**getgrent**, **getgrgid**() and **getgrnam**() each return pointers to an object with the following structure containing the broken-out fields of a line in the group file. Each line contains a "group" structure, defined in the **<grp.h>** header file.

```
struct  group {
        char    *gr_name;
        char    *gr_passwd;
        int     gr_gid;
        char    **gr_mem;
};
```

The members of this structure are:

| | |
|---|---|
| **gr_name** | The name of the group. |
| **gr_passwd** | The encrypted password of the group. |
| **gr_gid** | The numerical group ID. |
| **gr_mem** | A NULL-terminated array of pointers to the individual member names. |

**getgrent**() when first called returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. **getgrgid**() searches from the beginning of the file until a numerical group ID matching **gid** is found and returns a pointer to the particular structure in which it was found. **getgrnam**() searches from the beginning of the file until a group name matching *name* is found and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to **getgrent**() has the effect of rewinding the group file to allow repeated searches. **endgrent**() may be called to close the group file when processing is complete.

**fgetgrent**() returns a pointer to the next group structure in the stream *f*, which must refer to an open file in the same format as the group file **/etc/group**.

DIAGNOSTICS

A NULL pointer is returned on EOF or error.

FILES

**/etc/group**

SEE ALSO

**getlogin**(3), **getpwent**(3), **group**(5), **ypserv**(8)

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

Unlike the corresponding routines for passwords (see **getpwent**(3)), which always search the entire file, these routines start searching from the current file location.

WARNING

The above routines use **<stdio.h>**, which increases the size of programs, not otherwise using standard I/O, more than might be expected.

## NAME

gethostent, gethostbyaddr, gethostbyname, sethostent, endhostent – get network host entry

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

struct hostent *gethostent()

struct hostent *gethostbyname(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr; int len, type;

sethostent(stayopen)
int stayopen
endhostent()
```

## DESCRIPTION

**gethostent**, **gethostbyname**, and **gethostbyaddr()** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network host data base, **/etc/hosts**.

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* address type */
        int     h_length;       /* length of address */
        char    *h_addr;        /* address */
};
```

The members of this structure are:

| | |
|---|---|
| **h_name** | Official name of the host. |
| **h_aliases** | A zero terminated array of alternate names for the host. |
| **h_addrtype** | The type of address being returned; currently always **AF_INET.** |
| **h_length** | The length, in bytes, of the address. |
| **h_addr** | A pointer to the network address for the host. Host addresses are returned in network byte order. |

**gethostent()** reads the next line of the file, opening the file if necessary.

**sethostent()** opens and rewinds the file. If the *stayopen* flag is non-zero, the host data base will not be closed after each call to **gethostent()** (either directly, or indirectly through one of the other "gethost" calls).

**endhostent()** closes the file.

**gethostbyname()** and **gethostbyaddr()** sequentially search from the beginning of the file until a matching host name or host address is found, or until end-of-file is encountered. Host addresses are supplied in network order.

## FILES

**/etc/hosts**

## SEE ALSO

**hosts**(5), **ypserv**(8)

## DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

**BUGS**

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

NAME

getlogin – get login name

SYNOPSIS

**char *getlogin( )**

DESCRIPTION

**getlogin( )** returns a pointer to the login name as found in **/etc/utmp**. It may be used in conjunction with **getpwnam** to locate the correct password file entry when the same user ID is shared by several login names.

If **getlogin( )** is called within a process that is not attached to a terminal, or if there is no entry in **/etc/utmp** for the process's terminal, it returns a NULL pointer. The correct procedure for determining the login name is to call *cuserid*, or to call **getlogin( )** and, if it fails, to call **getpwuid(getuid( ))**.

FILES

**/etc/utmp**

SEE ALSO

**cuserid**(3S), **getpwent**(3), **utmp**(5)

DIAGNOSTICS

Returns a NULL pointer if the name is not found.

BUGS

The return values point to static data whose content is overwritten by each call.

**getlogin( )** does not work for processes running under a **pty** (for example, emacs shell buffers, or shell tools) unless the program ''fakes'' the login name in the **/etc/utmp** file.

NAME

getmntent, setmntent, addmntent, endmntent, hasmntopt – get file system descriptor file entry

SYNOPSIS

```
#include <stdio.h>
#include <mntent.h>

FILE *setmntent(filep, type)
char *filep;
char *type;

struct mntent *getmntent(filep)
FILE *filep;

int addmntent(filep, mnt)
FILE *filep;
struct mntent *mnt;

char *hasmntopt(mnt, opt)
struct mntent *mnt;
char *opt;

int endmntent(filep)
FILE *filep;
```

DESCRIPTION

These routines replace the **getfsent()** routines for accessing the file system description file **/etc/fstab**. They are also used to access the mounted file system description file **/etc/mtab**.

**setmntent()** opens a file system description file and returns a file pointer which can then be used with **getmntent**, **addmntent**, or **endmntent**. The *type* argument is the same as in **fopen**(3). **getmntent()** reads the next line from *filep* and returns a pointer to an object with the following structure containing the broken-out fields of a line in the filesystem description file, **<mntent.h>**. The fields have meanings described in **fstab**(5).

```
struct mntent {
        char    *mnt_fsname;   /* file system name */
        char    *mnt_dir;      /* file system path prefix */
        char    *mnt_type;     /* 4.2, nfs, swap, or xx */
        char    *mnt_opts;     /* ro, quota, etc. */
        int     mnt_freq;      /* dump frequency, in days */
        int     mnt_passno;    /* pass number on parallel fsck */
};
```

**addmntent()** adds the **mntent** structure *mnt* to the end of the open file *filep*. Note: *filep* has to be opened for writing if this is to work. **hasmntopt()** scans the **mnt_opts** field of the **mntent** structure *mnt* for a substring that matches *opt*. It returns the address of the substring if a match is found, 0 otherwise. **endmntent()** closes the file.

FILES

**/etc/fstab**
**/etc/mtab**

SEE ALSO

**fopen**(3S), **getfsent**(3), **fstab**(5)

DIAGNOSTICS

NULL pointer (0) returned on EOF or error.

BUGS

The returned **mntent** structure points to static information that is overwritten in each call.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get network entry

SYNOPSIS

**#include <netdb.h>**

**struct netent *getnetent( )**

**struct netent *getnetbyname(name)**
**char *name;**

**struct netent *getnetbyaddr(net, type)**
**long net;**
**int type;**

**setnetent**(stayopen)
int stayopen;

**endnetent( )**

DESCRIPTION

**getnetent, getnetbyname,** and **getnetbyaddr( )** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network data base, **/etc/networks.**

```
struct  netent {
        char    *n_name;        /* official name of net */
        char    **n_aliases;    /* alias list */
        int     n_addrtype;     /* net number type */
        long    n_net;          /* net number */
};
```

The members of this structure are:

**n_name** The official name of the network.

**n_aliases** A zero terminated list of alternate names for the network.

**n_addrtype** The type of the network number returned; currently only **AF_INET.**

**n_net** The network number. Network numbers are returned in machine byte order.

**getnetent( )** reads the next line of the file, opening the file if necessary.

**getnetent( )** opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **getnetent( )** (either directly, or indirectly through one of the other "getnet" calls).

**endnetent( )** closes the file.

**getnetbyname( )** and **getnetbyaddr( )** sequentially search from the beginning of the file until a matching net name or net address and type is found, or until end-of-file is encountered. Network numbers are supplied in host order.

FILES

**/etc/networks**

SEE ALSO

**networks**(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

Only Internet network numbers are currently understood.

NAME

getnetgrent, setnetgrent, endnetgrent, innetgr – get network group entry

SYNOPSIS

**innetgr(netgroup, machine, user, domain)**
**char *netgroup, *machine, *user, *domain;**

**setnetgrent(netgroup)**
**char *netgroup**

**endnetgrent( )**

**getnetgrent(machinep, userp, domainp)**
**char **machinep, **userp, **domainp;**

DESCRIPTION

**inngetgr** returns 1 or 0, depending on whether *netgroup* contains the machine, user, domain triple as a member. Any of the three strings *machine*, *user*, or *domain* can be NULL, in which case it signifies a wild card.

**getnetgrent( )** returns the next member of a network group. After the call, *machinep* will contain a pointer to a string containing the name of the machine part of the network group member, and similarly for *userp* and *domainp*. If any of *machinep*, *userp* or *domainp* is returned as a NULL pointer, it signifies a wild card. **getnetgrent( )** will use **malloc**(3) to allocate space for the name. This space is released when a **endnetgrent( )** call is made. **getnetgrent( )** returns 1 if it succeeding in obtaining another member of the network group, 0 if it has reached the end of the group.

**getnetgrent( )** establishes the network group from which **getnetgrent( )** will obtain members, and also restarts calls to **getnetgrent( )** from the beginning of the list. If the previous **setnetgrent( )** call was to a different network group, a **endnetgrent( )** call is implied. **endnetgrent( )** frees the space allocated during the **getnetgrent( )** calls.

FILES

**/etc/netgroup**

NAME

getopt, optarg, optind – get option letter from argument vector

SYNOPSIS

```
int getopt(argc, argv, optstring)
int argc;
char **argv;
char *optstring;

extern char *optarg;
extern int optind, opterr;
```

DESCRIPTION

**getopt()** returns the next option letter in *argv* that matches a letter in *optstring*. *optstring* must contain the option letters the command using **getopt()** will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

*optarg* is set to point to the start of the option argument on return from **getopt**.

**getopt()** places in **optind()** the *argv* index of the next argument to be processed. **optind()** is external and is initialized to **1** before the first call to **getopt**.

When all options have been processed (that is, up to the first non-option argument), **getopt()** returns **–1**. The special option "—" may be used to delimit the end of the options; when it is encountered, **–1** will be returned, and "—" will be skipped.

DIAGNOSTICS

**getopt()** prints an error message on the standard error and returns a question mark (**?**) when it encounters an option letter not included in *optstring* or no option-argument after an option that expects one. This error message may be disabled by setting **opterr** to **0**.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the option **o**, which requires an option argument:

```
main(argc, argv)
int argc;
char **argv;
{
        int c;
        extern char *optarg;
        extern int optind;
        .
        .
        .
        while ((c = getopt(argc, argv, "abo:")) != -1)
                switch (c) {
                case 'a':
                        if (bflg)
                                errflg++;
                        else
                                aflg++;
                        break;
                case 'b':
                        if (aflg)
                                errflg++;
                        else
                                bproc ();
                        break;
```

```
                        case 'o':
                                ofile = optarg;
                                break;
                        case '?':
                                errflg++;
                        }
                if (errflg) {
                        (void)fprintf(stderr, "usage: . . . ");
                        exit (2);
                }
                for (; optind < argc; optind++) {
                        if (access(argv[optind], 4)) {
                .
                .
                .
        }
```

SEE ALSO

**getopts**(1)

WARNING

Changing the value of the variable **optind**, or calling **getopt()** with different values of *argv*, may lead to unexpected results.

NAME

getpass – read a password

SYNOPSIS

**char *getpass(prompt)**
**char *prompt;**

DESCRIPTION

**getpass( )** reads up to a NEWLINE or EOF from the file **/dev/tty**, or if that cannot be opened, from the standard input, after prompting with the NULL-terminated string *prompt* and disabling echoing. A pointer is returned to a NULL-terminated string of at most 8 characters. An interrupt will terminate input and send an interrupt signal to the calling program before returning.

FILES

**/dev/tty**

SEE ALSO

**crypt**(3), **getpass**(3V)

WARNING

The above routine uses **<stdio.h>**, which increases the size of programs not otherwise using standard I/O, more than might be expected.

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocol entry

SYNOPSIS

```
#include <netdb.h>

struct protoent *getprotoent( )

struct protoent *getprotobyname(name)
char *name;

struct protoent *getprotobynumber(proto)
int proto;

setprotoent(stayopen)
int stayopen;

endprotoent( )
```

DESCRIPTION

**getprotoent**, **getprotobyname**, and **getprotobynumber( )** each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol data base, **/etc/protocols**.

```
struct  protoent {
        char    *p_name;        /* official name of protocol */
        char    **p_aliases;    /* alias list */
        int     p_proto;        /* protocol number */
};
```

The members of this structure are:

| | |
|---|---|
| **p_name** | The official name of the protocol. |
| **p_aliases** | A zero terminated list of alternate names for the protocol. |
| **p_proto** | The protocol number. |

**getprotoent( )** reads the next line of the file, opening the file if necessary.

**getprotoent( )** opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **getprotoent( )** (either directly, or indirectly through one of the other "getproto" calls).

**endprotoent( )** closes the file.

**getprotobyname( )** and **getprotobynumber( )** sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until end-of-file is encountered.

FILES

**/etc/protocols**

SEE ALSO

**protocols**(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

NAME

getpw – get name from uid

SYNOPSIS

**getpw(uid, buf)**
**char *buf;**

DESCRIPTION

**Getpw is made obsolete by getpwent(3).**

**getpw( )** searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is NULL-terminated.

FILES

**/etc/passwd**

SEE ALSO

**getpwent**(3), **passwd**(5)

DIAGNOSTICS

Non-zero return on error.

NAME

getpwaent, getpwanam, setpwaent, endpwaent, fgetpwaent – get password adjunct file entry

SYNOPSIS

```
#include <sys/types.h>
#include <sys/label.h>
#include <sys/audit.h>
#include <pwdadj.h>

struct passwd_adjunct *getpwaent()

struct passwd_adjunct *getpwanam(name)
char *name;

struct passwd_adjunct *fgetpwaent(f)
FILE *f;

void setpwaent()

void endpwaent()
```

DESCRIPTION

Both **getpwaent()** and **getpwanam()** return a pointer to an object with the following structure containing the broken-out fields of a line in the password adjunct file. Each line in the file contains a **passwd_adjunct** structure, declared in the **<pwdadj.h>** header file:

```
struct  passwd_adjunct {
char        *pwa_name;
char        *pwa_passwd;
blabel_t     pwa_minimum;
blabel_t     pwa_maximum;
blabel_t     pwa_def;
audit_state_t  pwa_au_always;
audit_state_t  pwa_au_never;
int         pwa_version;
};
```

When first called, **getpwaent()** returns a pointer to a **passwd_adjunct** structure describing data from the first line in the file. Thereafter, it returns a pointer to a **passwd_adjunct** structure describing data from the next line in the file. So successive calls can be used to search the entire file.

**getpwanam()** searches from the beginning of the file until it finds a login name matching *name*, then returns a pointer to the particular structure in which it was found.

Calling **getpwaent()** rewinds the password adjunct file to allow repeated searches. Calling **endpwaent()** closes the password adjunct file when processing is complete.

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

SEE ALSO

**getpwent**(3), **getgrent**(3), **passwd.adjunct**(5), **ypserv**(8)

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, setpwfile, fgetpwent – get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent()

int endpwent()

setpwfile(name)
char *name;

struct passwd *fgetpwent(f)
FILE *f;
```

DESCRIPTION

**getpwent**, **getpwuid**() and **getpwnam**() each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file. Each line in the file contains a "passwd" structure, declared in the **<pwd.h>** header file:

```
struct   passwd { /* see getpwent(3) */
         char    *pw_name;
         char    *pw_passwd;
         int     pw_uid;
         int     pw_gid;
         int     pw_quota;
         char    *pw_comment;
         char    *pw_gecos;
         char    *pw_dir;
         char    *pw_shell;
};
struct passwd *getpwent(), *getpwuid(), *getpwnam();
```

This structure is declared in **<pwd.h>** so it is not necessary to redeclare it.

The fields **pw_quota** and **pw_comment** are unused; the others have meanings described in **passwd**(5). When first called, **getpwent**() returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. **getpwuid**() searches from the beginning of the file until a numerical user ID matching *uid* is found and returns a pointer to the particular structure in which it was found. **getpwnam**() searches from the beginning of the file until a login name matching *name* is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to **getpwent**() has the effect of rewinding the password file to allow repeated searches. **endpwent**() may be called to close the password file when processing is complete.

**setpwfile**() changes the default password file to *name* thus allowing alternate password files to be used. Note: it does *not* close the previous file. If this is desired, **endpwent**() should be called prior to it.

**fgetpwent**() returns a pointer to the next passwd structure in the stream *f*, which matches the format of the password file **/etc/passwd**.

FILES

**/etc/passwd**

SEE ALSO

**getgrent**(3), **getlogin**(3), **getpwent**(3V), **passwd**(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

NAME

getrpcent, getrpcbyname, getrpcbynumber – get RPC entry

SYNOPSIS

```
#include <netdb.h>

struct rpcent *getrpcent( )

struct rpcent *getrpcbyname(name)
char *name;

struct rpcent *getrpcbynumber(number)
int number;

setrpcent (stayopen)
int stayopen

endrpcent ( )
```

DESCRIPTION

**getrpcent, getrpcbyname,** and **getrpcbynumber( )** each return a pointer to an object with the following structure containing the broken-out fields of a line in the rpc program number data base, **/etc/rpc.**

```
struct  rpcent {
        char    *r_name;        /* name of server for this rpc program */
        char    **r_aliases;    /* alias list */
        long    r_number;       /* rpc program number */
};
```

The members of this structure are:

| | |
|---|---|
| **r_name** | The name of the server for this rpc program. |
| **r_aliases** | A zero terminated list of alternate names for the rpc program. |
| **r_number** | The rpc program number for this service. |

**getrpcent( )** reads the next line of the file, opening the file if necessary.

**getrpcent( )** opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **getrpcent( )** (either directly, or indirectly through one of the other "getrpc" calls).

**endrpcent** closes the file.

**getrpcbyname( )** and **getrpcbynumber( )** sequentially search from the beginning of the file until a matching rpc program name or program number is found, or until end-of-file is encountered.

FILES

**/etc/rpc**

SEE ALSO

**rpc**(5), **rpcinfo**(8C), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

gets, fgets – get a string from a stream

SYNOPSIS

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;
```

DESCRIPTION

**gets( )** reads characters from the standard input stream, **stdin**, into the array pointed to by *s*, until a NEWLINE character is read or an EOF condition is encountered. The NEWLINE character is discarded and the string is terminated with a NULL character. **gets( )** returns its argument.

**fgets( )** reads characters from the stream into the array pointed to by *s*, until $n-1$ characters are read, a NEWLINE character is read and transferred to *s*, or an EOF condition is encountered. The string is then terminated with a NULL character. **fgets( )** returns its first argument.

SEE ALSO

**puts**(3S), **getc**(3S), **scanf**(3S), **fread**(3S), **ferror**(3S)

DIAGNOSTICS

If EOF is encountered and no characters have been read, no characters are transferred to *s* and a NULL pointer is returned. If a read error occurs, such as trying to use these functions on a file that has not been opened for reading, a NULL pointer is returned. Otherwise *s* is returned.

NAME

getservent, getservbyport, getservbyname, setservent, endservent – get service entry

SYNOPSIS

```
#include <netdb.h>

struct servent *getservent( )

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen;

endservent( )
```

DESCRIPTION

**getservent**, *getservbyname*, and *getservbyport* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, **/etc/services**.

```
struct  servent {
        char    *s_name;        /* official name of service */
        char    **s_aliases;    /* alias list */
        int     s_port;         /* port service resides at */
        char    *s_proto;       /* protocol to use */
};
```

The members of this structure are:

| | |
|---|---|
| **s_name** | The official name of the service. |
| **s_aliases** | A zero terminated list of alternate names for the service. |
| **s_port** | The port number at which the service resides. Port numbers are returned in network byte order. |
| **s_proto** | The name of the protocol to use when contacting the service. |

**getservent( )** reads the next line of the file, opening the file if necessary.

**getservent( )** opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to **getservent( )** (either directly, or indirectly through one of the other "getserv" calls).

**endservent( )** closes the file.

**getservbyname( )** and **getservbyport( )** sequentially search from the beginning of the file until a matching protocol name or port number is found, or until end-of-file is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

**/etc/services**

SEE ALSO

**getprotoent**(3N), **services**(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32 bit quantity is probably naive.

NAME

getttyent, getttynam, setttyent, endttyent – get ttytab file entry

SYNOPSIS

```
#include <ttyent.h>

struct ttyent *getttyent()

struct ttyent *getttynam(name)
char *name;

setttyent()

endttyent()
```

DESCRIPTION

**getttyent()** and **getttynam()** each return a pointer to an object with the following structure containing the broken-out fields of a line from the tty description file.

```
struct  ttyent {
        char    *ty_name;       /* terminal device name */
        char    *ty_getty;      /* command to execute, usually getty */
        char    *ty_type;       /* terminal type for termcap (3X) */
        int     ty_status;      /* status flags (see below for defines) */
        char    *ty_window;     /* command to start up window manager */
        char    *ty_comment;    /* usually the location of the terminal */
};
#define TTY_ON          0x1     /* enable logins (startup getty) */
#define TTY_SECURE      0x2     /* allow root to login */
```

**ty_name** is the name of the character-special file in the directory **/dev**. For various reasons, it must reside in the directory **/dev**.

**ty_getty** is the command (usually **getty**(8)) which is invoked by **init** to initialize tty line characteristics. In fact, any arbitrary command can be used; a typical use is to initiate a terminal emulator in a window system.

**ty_type** is the name of the default terminal type connected to this tty line. This is typically a name from the **termcap**(5) data base. The environment variable TERM is initialized with this name by **getty**(8) or **login**(1).

**ty_status** is a mask of bit fields which indicate various actions to be allowed on this tty line. The following is a description of each flag.

TTY_ON
: Enables logins (that is, **init**(8) will start the specified "getty" command on this entry).

TTY_SECURE
: Allows root to login on this terminal. Note: **TTY_ON** must be included for this to be useful.

**ty_window** is the command to execute for a window system associated with the line. The window system will be started before the command specified in the **ty_getty** entry is executed. If none is specified, this will be NULL.

**ty_comment** is the trailing comment field, if any; a leading delimiter and white space will be removed.

**getttyent()** reads the next line from the **ttytab** file, opening the file if necessary; **getttyent()** rewinds the file; **endttyent()** closes it.

**getttynam()** searches from the beginning of the file until a matching *name* is found (or until EOF is encountered).

**FILES**

**/etc/ttytab**

**SEE ALSO**

**login**(1), **ttyslot**(3), **ttyslot**(3V), **gettytab**(5), **ttytab**(5), **termcap**(5), **getty**(8), **init**(8)

**DIAGNOSTICS**

NULL pointer (0) returned on EOF or error.

**BUGS**

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getusershell, setusershell, endusershell – get legal user shells

SYNOPSIS

**char *getusershell()**

**setusershell()**

**endusershell()**

DESCRIPTION

**getusershell()** returns a pointer to a legal user shell as defined by the system manager in the file **/etc/shells**. If **/etc/shells** does not exist, the two standard system shells **/usr/bin/sh** and **/usr/bin/csh** are returned.

**getusershell()** reads the next line (opening the file if necessary); **setusershell()** rewinds the file; **endusershell()** closes it.

FILES

**/etc/shells**
**/usr/bin/csh**

DIAGNOSTICS

The routine **getusershell()** returns a NULL pointer (0) on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getwd – get current working directory pathname

SYNOPSIS

```
#include <sys/param.h>

char *getwd(pathname)
char pathname[MAXPATHLEN];
```

DESCRIPTION

**getwd( )** copies the absolute pathname of the current working directory to *pathname* and returns a pointer to the result.

DIAGNOSTICS

**getwd( )** returns zero and places a message in *pathname* if an error occurs.

FILES

| | |
|---|---|
| **/tmp/.getwd** | It exists for the sole purpose of the **getwd( )** library routine; no other software should depend on its existence or contents. |

NAME

hsearch, hcreate, hdestroy – manage hash search tables

SYNOPSIS

```
#include <search.h>

ENTRY *hsearch (item, action)
ENTRY item;
ACTION action;

int hcreate (nel)
unsigned nel;

void hdestroy ( )
```

DESCRIPTION

**hsearch**( ) is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. *item* is a structure of type ENTRY (defined in the **<search.h>** header file) containing two pointers: *item.key* points to the comparison key, and *item.data* points to any other data to be associated with that key. (Pointers to types other than character should be cast to pointer-to-character.) *action* is a member of an enumeration type ACTION indicating the disposition of the entry if it cannot be found in the table. ENTER indicates that the item should be inserted in the table at an appropriate point. FIND indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a NULL pointer.

**hcreate**( ) allocates sufficient space for the table, and must be called before **hsearch**( ) is used. *nel* is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

**hdestroy**( ) destroys the search table, and may be followed by another call to **hcreate**.

NOTES

**hsearch**( ) uses **open addressing** with a *multiplicative* hash function.

EXAMPLE

The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.

```
#include <stdio.h>
#include <search.h>
struct info {               /* this is the info stored in the table */
        int age, room;  /* other than the key. */
};
#define
NUM_EMPL    5000    /* # of elements in search table */
main( )
{
        /* space to store strings */
        char string_space[NUM_EMPL*20];
        /* space to store employee info */
        struct info info_space[NUM_EMPL];
        /* next avail space in string_space */
        char *str_ptr = string_space;
        /* next avail space in info_space */
        struct info *info_ptr = info_space;
        ENTRY item, *found_item, *hsearch( );
        /* name to look for in table */
        char name_to_find[30];
        int i = 0;
        /* create table */
```

```
        (void) hcreate(NUM_EMPL);
        while (scanf("%s%d%d", str_ptr, &info_ptr->age,
            &info_ptr->room) !=
EOF && i++ <
NUM_EMPL) {
                /* put info in structure, and structure in item */
                item.key = str_ptr;
                item.data = (char *)info_ptr;
                str_ptr += strlen(str_ptr) + 1;
                info_ptr++;
                /* put item into table */
                (void) hsearch(item,
ENTER);
        }
        /* access table */
        item.key = name_to_find;
        while (scanf("%s", item.key) != EOF) {
            if ((found_item = hsearch(item,
FIND)) != NULL) {
                /* if item is in the table */
                 (void)printf("found %s, age = %d, room = %d\n",
                        found_item->key,
                        ((struct info *)found_item->data)->age,
                        ((struct info *)found_item->data)->room);
            } else {
                 (void)printf("no such employee %s\n",
                        name_to_find)
            }
        }
}
```

SEE ALSO

**bsearch**(3), **lsearch**(3), **malloc**(3), **string**(3), **tsearch**(3)

DIAGNOSTICS

**hsearch**() returns a NULL pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

**hcreate**() returns zero if it cannot allocate sufficient space for the table.

WARNING

**hsearch**() and **hcreate**() use **malloc**(3) to allocate space.

BUGS

Only one hash search table may be active at any given time.

NAME

inet inet_addr, inet_network, inet_makeaddr, inet_lnaof, inet_netof, inet_ntoa – Internet address manipulation

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long
inet_addr(cp)
char *cp;

inet_network(cp)
char *cp;

struct in_addr
inet_makeaddr(net, lna)
int net, lna;

inet_lnaof(in)
struct in_addr in;

inet_netof(in)
struct in_addr in;

char *
inet_ntoa(in)
struct in_addr in;
```

DESCRIPTION

The routines **inet_addr()** and **inet_network()** each interpret character strings representing numbers expressed in the Internet standard '.' notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine **inet_makeaddr()** takes an Internet network number and a local network address and constructs an Internet address from it. The routines **inet_netof()** and **inet_lnaof()** break apart Internet host addresses, returning the network number and local network address part, respectively.

The routine **inet_ntoa()** returns a pointer to a string in the base 256 notation "d.d.d.d" described below.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES

Values specified using the '.' notation take one of the following forms:

**a.b.c.d**
**a.b.c**
**a.b**
**a**

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note: when an Internet address is viewed as a 32-bit integer quantity on Sun386i systems, the bytes referred to above appear as **d.c.b.a**. That is, Sun386i bytes are ordered from right to left.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as "128.net.host".

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as "net.host".

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as "parts" in a '.' notation may be decimal, octal, or hexadecimal, as specified in the C language (that is, a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO

**gethostent**(3N), **getnetent**(3N), **hosts**(5), **networks**(5),

DIAGNOSTICS

The value −1 is returned by **inet_addr()** and **inet_network()** for malformed requests.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed.

The return value from **inet_ntoa()** points to static information which is overwritten in each call.

NAME

initgroups – initialize group access list

SYNOPSIS

**initgroups(name, basegid)**
**char *name;**
**int basegid;**

DESCRIPTION

**initgroups()** reads through the group file and sets up, using the **setgroups** call (see **getgroups**(2)), the group access list for the user specified in *name*. The **basegid** is automatically included in the groups list. Typically this value is given as the group number from the password file.

FILES

**/etc/group**

SEE ALSO

**getgroups**(2), **getgrent**(3)

DIAGNOSTICS

**initgroups()** returns –1 if it was not invoked by the super-user.

BUGS

**initgroups()** uses the routines based on **getgrent**(3). If the invoking program uses any of these routines, the group structure will be overwritten in the call to **initgroups.**

NAME

insque, remque – insert/remove element from a queue

SYNOPSIS

```
struct qelem {
        struct  qelem *q_forw;
        struct  qelem *q_back;
        char    q_data[];
};

insque(elem, pred)
struct qelem *elem, *pred;

remque(elem)
struct qelem *elem;
```

DESCRIPTION

**insque()** and **remque()** manipulate queues built from doubly linked lists. Each element in the queue must be in the form of "struct qelem". **insque()** inserts *elem* in a queue immediately after *pred*; **remque()** removes an entry *elem* from a queue.

NAME

issecure – indicates whether system is running secure

SYNOPSIS

**int issecure()**

DESCRIPTION

This function tells whether the system has been configured to run in secure mode. It returns 0 if the system is not running secure, and non-zero if the system is running secure.

NAME

kvm_getu, kvm_getcmd – get the u-area or invocation arguments for a process

SYNOPSIS

```
#include <kvm.h>
#include <sys/param.h>
#include <sys/user.h>
#include <sys/proc.h>

struct user *kvm_getu(kd, proc)
kvm_t *kd;
struct proc *proc;

int kvm_getcmd(kd, proc, u, arg, env)
kvm_t *kd;
struct proc *proc;
struct user *u;
char ***arg;
char ***env;
```

DESCRIPTION

**kvm_getu()** reads the u-area of the process specified by *proc* to an area of static storage associated with *kd* and returns a pointer to it. Subsequent calls to **kvm_getu()** will overwrite the static u-area.

*kd* is a pointer to a kernel identifier returned by **kvm_open**(3K). *proc* is a pointer to a copy (in the current process' address space) of a *proc* structure (obtained, for instance, by a prior **kvm_nextproc**(3K) call).

**kvm_getcmd()** constructs a list of string pointers that represent the command arguments and environment that were used to initiate the process specified by *proc*.

*kd* is a pointer to a kernel identifier returned by **kvm_open**(3K). *u* is a pointer to a copy (in the current process' address space) of a *user* structure (obtained, for instance, by a prior **kvm_getu()** call). If *arg* is not NULL, then the command line arguments are formed into a NULL-terminated array of string pointers. The address of the first such pointer is returned in *arg*. If *env* is not NULL, then the environment is formed into a NULL-terminated array of string pointers. The address of the first of these is returned in *env*.

The pointers returned in *arg* and *env* refer to data allocated by **malloc**(3) and should be freed (by a call to **free** (see **malloc**(3)) when no longer needed. Both the string pointers and the strings themselves are deallocated when freed.

Since the environment and command line arguments may have been modified by the user process, there is no guarantee that it will be possible to reconstruct the original command at all. Thus, **kvm_getcmd()** will make the best attempt possible, returning –1 if the user process data is unrecognizable.

RETURN VALUE

**kvm_getu()** returns a NULL pointer if an error occurred.

**kvm_getcmd()** returns a value of 0 on successful completion. Otherwise –1 is returned.

SEE ALSO

**execve**(2), **kvm_nextproc**(3K), **kvm_open**(3K), **kvm_read**(3K), **malloc**(3)

NOTES

If **kvm_getcmd()** returns –1, the caller still has the option of using the command line fragment that is stored in the u-area.

NAME

kvm_getproc, kvm_nextproc, kvm_setproc – read system process structures

SYNOPSIS

```
#include <kvm.h>
#include <sys/param.h>
#include <sys/time.h>
#include <sys/proc.h>

struct proc *kvm_getproc(kd, pid)
kvm_t *kd;
int pid;

struct proc *kvm_nextproc(kd)
kvm_t *kd;

int kvm_setproc(kd)
kvm_t *kd;
```

DESCRIPTION

**kvm_nextproc()** may be used to sequentially read all of the system process structures from the kernel identified by *kd* (see **kvm_open**(3K)). Each call to **kvm_nextproc()** returns a pointer to the static memory area that contains a copy of the next valid process table entry. There is no guarantee that the data will remain valid across calls to **kvm_nextproc**, **kvm_setproc**, or **kvm_getproc**. Therefore, if the process structure must be saved, it should be copied to non-volatile storage.

For performance reasons, many implementations will cache a set of system process structures. Since the system state is liable to change between calls to **kvm_nextproc**, and since the cache may contain obsolete information, there is no guarantee that *every* process structure returned refers to an active process, nor is it certain that *all* processes will be reported.

**kvm_setproc()** rewinds the process list, enabling **kvm_nextproc()** to rescan from the beginning of the system process table. **kvm_setproc()** will always flush the process structure cache, allowing an application to re-scan the process table of a running system.

**kvm_getproc()** locates the *proc* structure of the process specified by *pid* and returns a pointer to it. **kvm_getproc()** does not interact with the process table pointer manipulated by **kvm_nextproc**, however, the restrictions regarding the validity of the data still apply.

RETURN VALUE

**kvm_getproc()** and **kvm_nextproc()** return a NULL pointer if an error has occurred.

**kvm_setproc()** returns a value of 0 on successful completion. Otherwise −1 is returned.

SEE ALSO

**kvm_getu**(3K), **kvm_open**(3K), **kvm_read**(3K)

## NAME

kvm_nlist – get entries from kernel symbol table

## SYNOPSIS

```
#include <kvm.h>
#include <nlist.h>

int kvm_nlist(kd, nl)
kvm_t *kd;
struct nlist *nl;
```

## DESCRIPTION

**kvm_nlist()** examines the symbol table from the kernel image identified by *kd* (see **kvm_open**(3K)) and selectively extracts a list of values and puts them in the array of **nlist()** structures pointed to by *nl*. The name list pointed to by **nl()** consists of an array of structures containing names, types and values. The *n_name* field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a NULL pointer (or a pointer to a NULL string) in the *n_name* field. For each entry in *nl*, if the named symbol is present in the kernel symbol table, its value and type are placed in the *n_value* and *n_type* fields. If a symbol cannot be located, the corresponding *n_type* field of **nl()** is set to zero.

## RETURN VALUE

Upon normal completion, **kvm_nlist()** returns the number of symbols that were not located in the symbol table. If an error occurs, **nlist()** returns –1 and sets all of the *n_type* fields in members of the array pointed to by **nl()** to zero.

## SEE ALSO

**kvm_open**(3K), **kvm_read**(3K), **nlist**(3), **a.out**(5)

NAME

kvm_open, kvm_close – specify a kernel to examine

SYNOPSIS

```
#include <kvm.h>
#include <fcntl.h>

kvm_t *kvm_open(namelist, corefile, swapfile, flag, errstr)
char *namelist, *corefile, *swapfile;
int flag;
char *errstr;

int kvm_close(kd)
kvm_t *kd;
```

DESCRIPTION

**kvm_open()** initializes a set of file descriptors to be used in subsequent calls to Kernel VM routines. It returns a pointer to a kernel identifier that must be used as the *kd* argument in subsequent Kernel VM function calls.

The *namelist* argument specifies an unstripped executable file whose symbol table will be used to locate various offsets in *corefile*. If *namelist* is NULL, the symbol table of the currently running kernel is used to determine offsets in the core image. In this case, it is up to the implementation to select an appropriate way to resolve symbolic references (for instance, using **/vmunix** as a default *namelist* file).

*corefile* specifies a file that contains an image of physical memory, for instance, a kernel crash dump file (see **savecore**(8)) or the special device **/dev/mem**. If *corefile* is NULL, the currently running kernel is accessed (using **/dev/mem** and **/dev/kmem**).

*swapfile* specifies a file that represents the swap device. If both *corefile* and *swapfile* are NULL, the swap device of the "currently running kernel" is accessed. Otherwise, if *swapfile* is NULL, **kvm_open()** may succeed but subsequent **kvm_getu**(3K) function calls may fail if the desired information is swapped out.

*flag* is used to specify read or write access for *corefile* and may have one of the following values:

| | |
|---|---|
| **O_RDONLY** | open for reading |
| **O_RDWR** | open for reading and writing |

*errstr* is used to control error reporting. If it is a NULL pointer, no error messages will be printed. If it is non-NULL, it is assumed to be the address of a string that will be used to prefix error messages generated by **kvm_open**. Errors are printed to **stderr**. A useful value to supply for *errstr* would be **argv**[0]. This has the effect of printing the process name in front of any error messages.

**kvm_close()** closes all file descriptors that were associated with *kd*. These files are also closed on **exit**(2) and **execve**(2). **kvm_close()** also resets the **proc** pointer associated with **kvm_nextproc**(3K) and flushes any cached kernel data.

RETURN VALUE

Upon successful completion, **kvm_open()** returns a non-NULL value that is suitable for use with subsequent Kernel VM function calls. If an error occurs, no files are opened and 0 is returned.

Upon successful completion, **kvm_close()** closes all file descriptors associated with *kd* and returns zero. Otherwise –1 is returned.

FILES

**/vmunix**
**/dev/kmem**
**/dev/mem**
**/dev/drum**

SEE ALSO

**execve**(2), **exit**(2), **kvm_getu**(3K), **kvm_nextproc**(3K), **kvm_nlist**(3K), **kvm_read**(3K), **savecore**(8)

NAME

kvm_read, kvm_write – copy data to or from a kernel image or running system

SYNOPSIS

```
#include <kvm.h>

int kvm_read(kd, addr, buf, nbytes)
kvm_t *kd;
unsigned long addr;
char *buf;
unsigned nbytes;

int kvm_write(kd, addr, buf, nbytes)
kvm_t *kd;
unsigned long addr;
char *buf;
unsigned nbytes;
```

DESCRIPTION

**kvm_read()** transfers data from the kernel image specified by *kd* (see **kvm_open**(3K)) to the address space of the process. *nbytes* bytes of data are copied from the kernel virtual address given by *addr* to the buffer pointed to by *buf*.

**kvm_write()** is like **kvm_read**, except that the direction of data transfer is reversed. In order to use this function, the **kvm_open**(3K) call that returned *kd* must have specified write access.

RETURN VALUE

Upon normal completion, the number of bytes successfully transferred is returned. Otherwise –1 is returned.

SEE ALSO

**kvm_getu**(3K), **kvm_nlist**(3K), **kvm_open**(3K)

NAME

ldahread – read the archive header of a member of a COFF archive file

SYNOPSIS

```
#include <stdio.h>
#include <ar.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldahread (ldptr, arhead)
LDFILE *ldptr;
ARCHDR *arhead;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

If TYPE(*ldptr*) is the archive file magic number, **ldahread** reads the archive header of the COFF file currently associated with *ldptr* into the area of memory beginning at *arhead*.

**ldahread** returns SUCCESS or FAILURE. **ldahread** will fail if TYPE(*ldptr*) does not represent an archive file, or if it cannot read the archive header.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldopen**(3X), **intro**(5), **ldfcn**(5)

NAME

ldclose, ldaclose – close a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldclose (ldptr)
LDFILE *ldptr;

int ldaclose (ldptr)
LDFILE *ldptr;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldopen**(3X) and **ldclose**() are designed to provide uniform access to both simple COFF object files and COFF object files that are members of archive files. Thus an archive of COFF files can be processed as if it were a series of simple COFF files.

If **TYPE**(*ldptr*) does not represent an archive file, **ldclose**() will close the file and free the memory allocated to the **LDFILE** structure associated with *ldptr*. If **TYPE**(*ldptr*) is the magic number of an archive file, and if there are any more files in the archive, **ldclose**() will reinitialize **OFFSET**(*ldptr*) to the file address of the next archive member and return **FAILURE**. The **LDFILE** structure is prepared for a subsequent **ldopen**(3X). In all other cases, **ldclose**() returns **SUCCESS**.

**ldaclose**() closes the file and frees the memory allocated to the **LDFILE** structure associated with *ldptr* regardless of the value of **TYPE**(*ldptr*). **ldaclose**() always returns **SUCCESS**. The function is often used in conjunction with *ldaopen*.

The program must be loaded with the object file access routine library **libld.a**.

*intro*(5) describes *INCDIR* and *LIBDIR*.

SEE ALSO

**fclose**(3S), **ldfcn**(3), **ldopen**(3X), **intro**(5)

## NAME

ldfcn – common object file access routines

## SYNOPSIS

**#include <stdio.h>**
**#include <filehdr.h>**
**#include <ldfcn.h>**

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

These routines are for reading COFF object files and archives containing COFF object files. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function **ldopen**(3X) allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

LDFILE *ldptr;

TYPE(ldptr) The file magic number used to distinguish between archive members and simple object files.

IOPTR(ldptr) The file pointer returned by *fopen* and used by the standard input/output functions.

OFFSET(ldptr) The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.

HEADER(ldptr) The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

(1) Functions that open or close an object file

**ldopen**(3X) and **ldaopen**(see **ldopen**(3X))
open a common object file
**ldclose**(3X) and **ldaclose**[see **ldclose**(3X)]
close a common object file

(2) Functions that read header or symbol table information

**ldahread**(3X)
read the archive header of a member of an archive file
**ldfhread**(3X)
read the file header of a common object file
**ldshread**(3X) and **ldnshread**[see **ldshread**(3X)]
read a section header of a common object file
**ldtbread**(3X)
read a symbol table entry of a common object file
**ldgetname**(3X)
retrieve a symbol name from a symbol table entry or from the string table

(3) Functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

**ldohseek**(3X)
seek to the optional file header of a common object file
**ldsseek**(3X) and **ldnsseek**[see **ldsseek**(3X)]
seek to a section of a common object file
**ldrseek**(3X) and **ldnrseek**[see **ldrseek**(3X)]
seek to the relocation information for a section of a common object file
**ldlseek**(3X) and **ldnlseek**[see **ldlseek**(3X)]
seek to the line number information for a section of a common object file
**ldtbseek**(3X)
seek to the symbol table of a common object file

(4) The unction **ldtbindex**(3X), which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages.

All the functions except **ldopen**(3X), **ldgetname**(3X), **ldtbindex**(3X) return either SUCCESS or **FAILURE**, both constants defined in **ldfcn.h**. **ldopen**(3X) and **ldaopen**[(see **ldopen**(3X)] both return pointers to an **LDFILE** structure.

Additional access to an object file is provided through a set of macros defined in **ldfcn.h**. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the **LDFILE** structure into a reference to its file descriptor field.

The following macros are provided:

GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)

The STROFFSET macro calculates the address of the string table. See the manual entries for the corresponding standard input/output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**fseek**(3S), **ldahread**(3X), **ldclose**(3X), **ldgetname**(3X), **ldfhread**(3X), **ldlread**(3X), **ldlseek**(3X), **ldohseek**(3X), **ldopen**(3X), **ldrseek**(3X), **ldlseek**(3X), **ldshread**(3X), **ldtbindex**(3X), **ldtbread**(3X), **ldtbseek**(3X), **stdio**(3S), **intro**(5)

WARNING

The macro **FSEEK** defined in the header file **ldfcn.h** translates into a call to the standard input/output function **fseek**(3S). **FSEEK** should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members.

NAME

ldfhread – read the file header of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldfhread (ldptr, filehead)
LDFILE *ldptr;
FILHDR *filehead;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldfhread()** reads the file header of the COFF file currently associated with *ldptr* into the area of memory beginning at *filehead*.

**ldfhread()** returns SUCCESS or FAILURE. **ldfhread()** will fail if it cannot read the file header.

In most cases the use of **ldfhread()** can be avoided by using the macro **HEADER(***ldptr***)** defined in **ldfcn.h** (see **ldfcn**(3)). The information in any field, *fieldname*, of the file header may be accessed using **HEADER(ldptr).fieldname**.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X)

NAME

ldgetname – retrieve symbol name for COFF file symbol table entry

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

char *ldgetname (ldptr, symbol)
LDFILE *ldptr;
SYMENT *symbol;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldgetname()** returns a pointer to the name associated with **symbol** as a string. The string is contained in a static buffer local to **ldgetname()** that is overwritten by each call to **ldgetname()**, and therefore must be copied by the caller if the name is to be saved.

**ldgetname()** can be used to retrieve names from object files without any backward compatibility problems. **ldgetname()** will return NULL (defined in **stdio.h**) for an object file if the name cannot be retrieved. This situation can occur:

- if the "string table" cannot be found,
- if not enough memory can be allocated for the string table,
- if the string table appears not to be a string table (for example, if an auxiliary entry is handed to **ldgetname()** that looks like a reference to a name in a nonexistent string table), or
- if the name's offset into the string table is past the end of the string table.

Typically, **ldgetname()** will be called immediately after a successful call to **ldtbread()** to retrieve the name associated with the symbol table entry filled by **ldtbread()**.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldtbread**(3X), **ldtbseek**(3X)

NAME

ldlread, ldlinit, ldlitem – manipulate line number entries of a COFF file function

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <linenum.h>
#include <ldfcn.h>

int ldlread(ldptr, fcnindx, linenum, linent)
LDFILE *ldptr;
long fcnindx;
unsigned short linenum;
LINENO *linent;

int ldlinit(ldptr, fcnindx)
LDFILE *ldptr;
long fcnindx;

int ldlitem(ldptr, linenum, linent)
LDFILE *ldptr;
unsigned short linenum;
LINENO *linent;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldlread()** searches the line number entries of the COFF file currently associated with *ldptr*. **ldlread()** begins its search with the line number entry for the beginning of a function and confines its search to the line numbers associated with a single function. The function is identified by *fcnindx*, the index of its entry in the object file symbol table. **ldlread()** reads the entry with the smallest line number equal to or greater than *linenum* into the memory beginning at *linent*.

**ldlinit()** and **ldlitem()** together perform exactly the same function as **ldlread()**. After an initial call to **ldlread()** or **ldlinit()**, **ldlitem()** may be used to retrieve a series of line number entries associated with a single function. **ldlinit()** simply locates the line number entries for the function identified by *fcnindx*. **ldlitem()** finds and reads the entry with the smallest line number equal to or greater than *linenum* into the memory beginning at **linent()**.

**ldlread()**, **ldlinit()**, and **ldlitem()** each return either SUCCESS or FAILURE. **ldlread()** will fail if there are no line number entries in the object file, if *fcnindx* does not index a function entry in the symbol table, or if it finds no line number equal to or greater than *linenum*. **ldlinit()** will fail if there are no line number entries in the object file or if *fcnindx* does not index a function entry in the symbol table. **ldlitem()** will fail if it finds no line number equal to or greater than *linenum*.

The programs must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldtbindex**(3X)

NAME

ldlseek, ldnlseek – seek to line number entries of a section of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldlseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnlseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldlseek**() seeks to the line number entries of the section specified by *sectindx* of the COFF file currently associated with *ldptr*.

**ldnlseek**() seeks to the line number entries of the section specified by *sectname*.

**ldlseek**() and **ldnlseek**() return SUCCESS or FAILURE. **ldlseek**() will fail if *sectindx* is greater than the number of sections in the object file; **ldnlseek**() will fail if there is no section name corresponding with **sectname*. Either function will fail if the specified section has no line number entries or if it cannot seek to the specified line number entries.

Note that the first section has an index of **one**.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldshread**(3X)

NAME

ldohseek – seek to the optional file header of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldohseek (ldptr)
LDFILE *ldptr;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldohseek()** seeks to the optional file header of the COFF file currently associated with *ldptr*.

**ldohsee()** returns SUCCESS or FAILURE. **ldohseek()** will fail if the object file has no optional header or if it cannot seek to the optional header.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldfhread**(3X)

NAME

ldopen, ldaopen – open a COFF file for reading

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

LDFILE *ldopen (filename, ldptr)
char *filename;
LDFILE *ldptr;

LDFILE *ldaopen (filename, oldptr)
char *filename;
LDFILE *oldptr;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldopen()** and **ldclose**(3X) are designed to provide uniform access to both simple object files and object files that are members of archive files. Thus an archive of COFF files can be processed as if it were a series of simple COFF files.

If *ldptr* has the value NULL, then **ldopen()** will open *filename* and allocate and initialize the LDFILE structure, and return a pointer to the structure to the calling program.

If *ldptr* is valid and if TYPE(*ldptr*) is the archive magic number, **ldopen()** will reinitialize the LDFILE structure for the next archive member of *filename*.

**ldopen()** and **ldclose**(3X) are designed to work in concert. *ldclose* will return FAILURE only when TYPE(*ldptr*) is the archive magic number and there is another file in the archive to be processed. Only then should **ldopen()** be called with the current value of *ldptr*. In all other cases, in particular whenever a new *filename* is opened, **ldopen()** should be called with a NULL *ldptr* argument.

The following is a prototype for the use of **ldopen()** and **ldclose**(3X).

```
/* for each filename to be processed */

ldptr = NULL;
do
{
        if ( (ldptr = ldopen(filename, ldptr)) != NULL )
        {
                /* check magic number */
                /* process the file */
        }
} while (ldclose(ldptr) == FAILURE );
```

If the value of *oldptr* is not NULL, **ldaopen()** will open *filename* anew and allocate and initialize a new LDFILE structure, copying the TYPE, OFFSET, and HEADER fields from *oldptr*. **ldaopen()** returns a pointer to the new LDFILE structure. This new pointer is independent of the old pointer, *oldptr*. The two pointers may be used concurrently to read separate parts of the object file. For example, one pointer may be used to step sequentially through the relocation information, while the other is used to read indexed symbol table entries.

Both **ldopen()** and **ldaopen()** open *filename* for reading. Both functions return NULL if *filename* cannot be opened, or if memory for the LDFILE structure cannot be allocated. A successful open does not insure that the given file is a COFF file or an archived object file.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**fopen**(3S), **ldclose**(3X), **ldfcn**(3)

NAME

ldrseek, ldnrseek – seek to relocation entries of a section of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldrseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnrseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldrseek**( ) seeks to the relocation entries of the section specified by *sectindx* of the COFF file currently associated with *ldptr*.

**ldnrseek**( ) seeks to the relocation entries of the section specified by *sectname*.

**ldrseek**( ) and **ldnrseek**( ) return SUCCESS or FAILURE. **ldrseek**( ) will fail if *sectindx* is greater than the number of sections in the object file; **ldnrseek**( ) will fail if there is no section name corresponding with *sectname*. Either function will fail if the specified section has no relocation entries or if it cannot seek to the specified relocation entries.

Note: the first section has an index of **one**.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldshread**(3X)

NAME

ldshread, ldnshread – read an indexed/named section header of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <scnhdr.h>
#include <ldfcn.h>

int ldshread (ldptr, sectindx, secthead)
LDFILE *ldptr;
unsigned short sectindx;
SCNHDR *secthead;

int ldnshread (ldptr, sectname, secthead)
LDFILE *ldptr;
char *sectname;
SCNHDR *secthead;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldshread()** reads the section header specified by *sectindx* of the COFF file currently associated with *ldptr* into the area of memory beginning at *secthead*.

**ldnshread()** reads the section header specified by *sectname* into the area of memory beginning at *secthead*.

**ldshread()** and **ldnshread()** return SUCCESS or FAILURE. **ldshread()** will fail if *sectindx* is greater than the number of sections in the object file; **ldnshread()** will fail if there is no section name corresponding with *sectname*. Either function will fail if it cannot read the specified section header.

Note: the first section header has an index of *one*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X)

NAME

ldsseek, ldnsseek – seek to an indexed/named section of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldsseek (ldptr, sectindx)
LDFILE *ldptr;
unsigned short sectindx;

int ldnsseek (ldptr, sectname)
LDFILE *ldptr;
char *sectname;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldsseek()** seeks to the section specified by *sectindx* of the COFF file currently associated with *ldptr*.

**ldnsseek()** seeks to the section specified by *sectname*.

**ldsseek()** and **ldnsseek()** return SUCCESS or FAILURE. **ldsseek()** will fail if *sectindx* is greater than the number of sections in the object file; **ldnsseek()** will fail if there is no section name corresponding with *sectname*. Either function will fail if there is no section data for the specified section or if it cannot seek to the specified section.

Note: the first section has an index of *one*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldshread**(3X)

## NAME

ldtbindex – compute the index of a symbol table entry of a COFF file

## SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

long ldtbindex (ldptr)
LDFILE *ldptr;
```

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

**ldtbindex()** returns the (**long**) index of the symbol table entry at the current position of the COFF file associated with *ldptr*.

The index returned by **ldtbindex()** may be used in subsequent calls to **ldtbread**(3X). However, since **ldtbindex()** returns the index of the symbol table entry that begins at the current position of the object file, if **ldtbindex()** is called immediately after a particular symbol table entry has been read, it will return the index of the next entry.

**ldtbindex()** will fail if there are no symbols in the object file, or if the object file is not positioned at the beginning of a symbol table entry.

Note that the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library **libld.a**.

## SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldtbread**(3X), **ldtbseek**(3X)

NAME

ldtbread – read an indexed symbol table entry of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <syms.h>
#include <ldfcn.h>

int ldtbread (ldptr, symindex, symbol)
LDFILE *ldptr;
long symindex;
SYMENT *symbol;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldtbread**() reads the symbol table entry specified by *symindex* of the COFF file currently associated with *ldptr* into the area of memory beginning at **symbol**.

**ldtbread**() returns SUCCESS or FAILURE. **ldtbread**() will fail if *symindex* is greater than or equal to the number of symbols in the object file, or if it cannot read the specified symbol table entry.

Note: the first symbol in the symbol table has an index of *zero*.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldtbseek**(3X), **ldgetname**(3X)

NAME

ldtbseek – seek to the symbol table of a COFF file

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

int ldtbseek (ldptr)
LDFILE *ldptr;
```

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**ldtbseek()** seeks to the symbol table of the COFF file currently associated with *ldptr*.

**ldtbseek()** returns SUCCESS or FAILURE. **ldtbseek()** will fail if the symbol table has been stripped from the object file, or if it cannot seek to the symbol table.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

**ldclose**(3X), **ldfcn**(3), **ldopen**(3X), **ldtbread**(3X)

NAME

lockf – advisory record locking on files

SYNOPSIS

```
#include <unistd.h>

#define F_ULOCK     0     /* Unlock a previously locked section */
#define F_LOCK      1     /* Lock a section for exclusive use */
#define F_TLOCK     2     /* Test and lock a section (non-blocking) */
#define F_TEST 3          /* Test section for other process' locks */

int lockf(fd, cmd, size)
int fd, cmd;
long size;
```

DESCRIPTION

**lockf()** may be used to test, apply, or remove an *advisory* record lock on the file associated with the open descriptor *fd*. (See **fcntl**(2V) for more information about advisory record locking.)

A lock is obtained by specifying a *cmd* parameter of **F_LOCK** or **F_TLOCK**. To unlock an existing lock, the **F_ULOCK** *cmd* is used. **F_TEST** is used to detect if a lock by another process is present on the specified segment.

**F_LOCK** and **F_TLOCK** requests differ only by the action taken if the lock may not be immediately granted. **F_TLOCK** returns a −1 by the function and sets **errno** to EAGAIN if the section is already locked by another process. **F_LOCK** will cause the process to sleep until the lock may be granted or a signal is caught.

*size* is the number of contiguous bytes to be locked or unlocked. The lock starts at the current file offset in the file and extends forward for a positive *size* or backward for a negative *size* (preceeding but not including the current offset). A segment need not be allocated to the file in order to be locked; however, a segment may not extend to a negative offset relative to the beginning of the file. If *size* is zero, the lock will extend from the current offset through the EOF. If such a lock starts at offset 0, then the entire file will be locked (regardless of future file extensions).

NOTES

The descriptor *fd* must have been opened with **O_WRONLY** or **O_RDWR** permission in order to establish locks with this function call.

All locks associated with a file for a given process are removed when the file is closed or the process terminates. Locks are not inherited by the child process in a **fork**(2) system call.

RETURN VALUE

Zero is returned on success, −1 on error, with an error code stored in **errno**.

ERRORS

**lockf()** will fail if one or more of the following are true:

| | |
|---|---|
| EBADF | *fd* is not a valid open descriptor. |
| EBADF | *cmd* is **F_LOCK** or **F_TLOCK** and the process does not have write permission on the file. |
| EAGAIN | *cmd* is **F_TLOCK** or **F_TEST** and the section is already locked by another process. |
| EINTR | *cmd* is **F_LOCK** and a signal interrupted the process while it was waiting for the lock to be granted. |
| ENOLCK | *cmd* is **F_LOCK**, **F_TLOCK**, or **F_ULOCK** and there are no more file lock entries available. |

SEE ALSO

**fcntl**(2V), **flock**(2), **fork**(2), **lockd**(8C)

BUGS

File locks obtained through the **lockf()** mechanism do not interact in any way with those acquired using **flock**(2). They do, however, work correctly with the locks claimed by **fcntl**(2V).

NAME

lsearch, lfind – linear search and update

SYNOPSIS

```
#include <stdio.h>
#include <search.h>

char *lsearch ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
        int (*compar)( );

char *lfind ((char *)key, (char *)base, nelp, sizeof(*key), compar)
unsigned *nelp;
int
        (*compar)( );
```

DESCRIPTION

**lsearch()** is a linear search routine generalized from Knuth (6.1) Algorithm S. It returns a pointer into a table indicating where a datum may be found. If the datum does not occur, it is added at the end of the table. **key()** points to the datum to be sought in the table. **base** points to the first element in the table. **nelp** points to an integer containing the current number of elements in the table. The integer is incremented if the datum is added to the table. **compar** is the name of the comparison function which the user must supply (**strcmp**, for example). It is called with two arguments that point to the elements being compared. The function must return zero if the elements are equal and non-zero otherwise.

**lfind()** is the same as **lsearch()** except that if the datum is not found, it is not added to the table. Instead, a NULL pointer is returned.

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLE

This fragment will read in ≤ TABSIZE strings of length ≤ ELSIZE and store them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <search.h>
#define
TABSIZE 50
#define
ELSIZE 120
        char line[ELSIZE], tab[TABSIZE][ELSIZE], *lsearch( );
        unsigned nel = 0;
        int strcmp( );
        . . .
        while (fgets(line,
        ELSIZE, stdin) != NULL &&
          nel < TABSIZE)
                (void) lsearch(line, (char *)tab, &nel, ELSIZE, strcmp);
        . . .
```

SEE ALSO

**bsearch**(3), **hsearch**(3), **tsearch**(3)

DIAGNOSTICS

If the searched for datum is found, both **lsearch**() and **lfind**() return a pointer to it. Otherwise, **lfind**() returns NULL and **lsearch**() returns a pointer to the newly added element.

BUGS

Undefined results can occur if there is not enough room in the table to add a new item.

NAME

malloc, free, realloc, calloc, cfree, memalign, valloc, alloca, malloc_debug, malloc_verify – memory allocator

SYNOPSIS

```
char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

cfree(ptr)
char *ptr;

char *memalign(alignment, size)
unsigned alignment;
unsigned size;

char *valloc(size)
unsigned size;

#include <alloca.h>
char *alloca(size)
int size;
```

DESCRIPTION

These routines provide a general-purpose memory allocation package. They maintain a table of free blocks for efficient allocation and coalescing of free storage. When there is no suitable space already free, the allocation routines call **sbrk**( ) (see **brk**(2)) to get more memory from the system.

Each of the allocation routines returns a pointer to space suitably aligned for storage of any type of object. Each returns a NULL pointer if the request cannot be completed (see DIAGNOSTICS).

**malloc**( ) returns a pointer to a block of at least *size* bytes, which is appropriately aligned. A NULL (0) pointer is returned if *size* is 0 or if *size* bytes of memory cannot be allocated.

**free**( ) releases a previously allocated block. Its argument is a pointer to a block previously allocated by **malloc, calloc, realloc, malloc,** or **memalign.**

**realloc**( ) changes the size of the block referenced by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes. For backwards compatibility, **realloc**( ) accepts a pointer to a block freed since the most recent call to **malloc, calloc, realloc, valloc,** or **memalign.** Note: using **realloc**( ) with a block freed *before* the most recent call to **malloc, calloc, realloc, valloc,** or **memalign** is an error.

**calloc**( ) uses **malloc**( ) to allocate space for an array of *nelem* elements of size *elsize*, initializes the space to zeros, and returns a pointer to the initialized block. The block can be freed with **free**( ) or **cfree.**

**memalign**( ) allocates *size* bytes on a specified alignment boundary, and returns a pointer to the allocated block. The value of the returned address is guaranteed to be an even multiple of *alignment*. Note: the value of *alignment* must be a power of two, and must be greater than or equal to the size of a word.

**valloc**(*size*) is equivalent to **memalign(getpagesize**( ), *size*).

**alloca**( ) allocates *size* bytes of space in the stack frame of the caller, and returns a pointer to the allocated block. This temporary space is automatically freed when the caller returns.

ERRORS

**malloc, calloc, realloc, valloc, memalign, cfree,** and **free()** will each fail if one or more of the following are true:

EINVAL — The requested allocation size is zero or an invalid argument was specified. The value of *ptr* passed to **free, cfree,** or **realloc()** must be a pointer to a block previously allocated by **malloc, calloc, realloc, valloc,** or **memalign.**

EINVAL — The allocation heap is found to have been corrupted. More detailed information may be obtained by enabling range checks using **malloc_debug.**

ENOMEM — *size* bytes of memory could not be allocated.

DIAGNOSTICS

More detailed diagnostics can be made available to programs using **malloc, calloc, realloc, valloc, memalign, cfree,** and **free,** by including a special relocatable object file at link time (see FILES). This file also provides routines for control of error handling and diagnosis, as defined below. Note: these routines are *not* defined in the standard library.

```
int malloc_debug(level)
int level;

int malloc_verify()
```

**malloc_debug()** sets the level of error diagnosis and reporting during subsequent calls to **malloc, calloc, realloc, valloc, memalign, cfree,** and **free.** The value of *level* is interpreted as follows:

Level 0 — **malloc, calloc, realloc, valloc, memalign, cfree,** and **free** behave the same as in the standard library.

Level 1 — The routines abort with a message to the standard error if errors are detected in arguments or in the heap. If a bad block is encountered, its address and size are included in the message.

Level 2 — Same as level 1, except that the entire heap is examined on every call to the above routines.

**malloc_debug()** returns the previous error diagnostic level. The default level is 1.

**malloc_verify()** attempts to determine if the heap has been corrupted. It scans all blocks in the heap (both free and allocated) looking for strange addresses or absurd sizes, and also checks for inconsistencies in the free space table. **malloc_verify()** returns 1 if all checks pass without error, and otherwise returns 0. The checks can take a significant amount of time, so it should not be used indiscriminately.

FILES

**/usr/lib/debug/malloc.o** — diagnostic versions of **malloc()** routines.

BUGS

**alloca()** is both machine- and compiler-dependent; its use is discouraged.

Since **realloc()** accepts a pointer to a block freed since the last call to **malloc, calloc, realloc, valloc,** or **memalign,** a degradation of performance results. The semantics of **free()** should be changed so that the contents of a previously freed block are undefined.

**SEE ALSO**

**brk**(2)

Stephenson, C.J., *Fast Fits*, in *Proceedings of the ACM 9th Symposium on Operating Systems*, **SIGOPS** *Operating Systems Review*, vol. 17, no. 5, October 1983.
*Core Wars*, in *Scientific American*, May 1984.

NAME

memory, memccpy, memchr, memcmp, memcpy, memset – memory operations

SYNOPSIS

```
#include <memory.h>

char *memccpy (s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr (s, c, n)
char *s;
int c, n;

int memcmp (s1, s2, n)
char *s1, *s2;
int n;

char *memcpy (s1, s2, n)
char *s1, *s2;
int n;

char *memset (s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate as efficiently as possible on memory areas (arrays of characters bounded by a count, not terminated by a NULL character). They do not check for the overflow of any receiving memory area.

**memccpy()** copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters have been copied, whichever comes first. It returns a pointer to the character after the copy of *c* in *s1*, or a NULL pointer if *c* was not found in the first *n* characters of *s2*.

**memchr()** returns a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s*, or a NULL pointer if *c* does not occur.

**memcmp()** compares its arguments, looking at the first *n* characters only, and returns an integer less than, equal to, or greater than 0, according as *s1* is lexicographically less than, equal to, or greater than *s2*.

**memcpy()** copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

**memset()** sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

NOTE

For user convenience, all these functions are declared in the **<memory.h>** header file.

BUGS

**memcmp()** uses native character comparison, which is signed on some machines and unsigned on other machines. Thus the sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

mktemp, mkstemp – make a unique file name

SYNOPSIS

**char *mktemp(template)**
**char *template;**

**mkstemp(template)**
**char *template;**

DESCRIPTION

**mktemp()** creates a unique file name, typically in a temporary filesystem, by replacing *template* with a unique file name, and always returns the address of *template*. The string in *template* should contain a file name with six trailing Xs; **mktemp()** replaces the Xs with a letter and the current process ID. The letter will be chosen so that the resulting name does not duplicate an existing file. **mkstemp()** makes the same replacement to the template but returns a file descriptor for the template file open for reading and writing. **mkstemp()** avoids the race between testing whether the file exists and opening it for use.

Notes:

- **mktemp()** and **mkstemp()** actually *change* the template string which you pass; this means that you cannot use the same template string more than once — you need a fresh template for every unique file you want to open.
- When **mktemp()** or **mkstemp()** are creating a new unique filename they check for the prior existence of a file with that name. This means that if you are creating more than one unique filename, it is bad practice to use the same root template for multiple invocations of **mktemp()** or **mkstemp()**.

SEE ALSO

**getpid**(2), **open**(2V), **tmpfile**(3S), **tmpnam**(3S)

DIAGNOSTICS

**mkstemp()** returns an open file descriptor upon success. It returns −1 if no suitable file could be created.

BUGS

It is possible to run out of letters.

## NAME

monitor, monstartup, moncontrol – prepare execution profile

## SYNOPSIS

```
#include <a.out.h>

monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[];

monstartup(lowpc, highpc)
int (*lowpc)(), (*highpc)();

moncontrol(mode)
```

## DESCRIPTION

There are two different forms of monitoring available. An executable program created by '**cc –p**' automatically includes calls for the **prof**(1) monitor, and includes an initial call with default parameters to its start-up routine **monstartup**. In this case, **monitor**( ) need not be called explicitly, except to gain fine control over **profil**(2) buffer allocation. An executable program created by '**cc –pg**' automatically includes calls for the **gprof**(1) monitor.

**monstartup**( ) is a high-level interface to **profil**(2). *lowpc* and *highpc* specify the address range that is to be sampled; the lowest address sampled is that of *lowpc* and the highest is just below *highpc*. **monstartup**( ) allocates space using **sbrk** (see **brk**(2)) and passes it to **monitor**( ) (as described below) to record a histogram of program-counter values, and calls to certain functions. Only calls to functions compiled with '**cc –p**' are recorded.

On Sun-2, Sun-3, and Sun-4 systems, an entire program can be profiled with:

```
extern etext();
...
monstartup(N_TXTOFF(0), etext);
```

On Sun386i systems, the equivalent code sequence is:

```
extern etext();
extern _start();
...
monstartup(_start, etext);
```

**etext** lies just above all the program text, see **end**(3).

To stop execution monitoring and post results to the file **mon.out**, use:

```
monitor(0);
```

**prof**(1) can then be used to examine the results.

**moncontrol**( ) is used to selectively control profiling within a program. This works with both **prof**(1) and **gprof**(1). Profiling begins when the program starts. To stop the collection of profiling statistics, use:

```
moncontrol(0)
```

To resume the collection of statistics, use:

```
moncontrol(1)
```

This allows you to measure the cost of particular functions. Note: an output file is be produced upon program exit, regardless of the state of **moncontrol**.

**monitor**( ) is a low level interface to **profil**(2). *lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* short integers. At most *nfunc* call counts can be kept.

For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. **monitor()** divides the buffer into space to record the histogram of program counter samples over the range *lowpc* to *highpc*, and space to record call counts of functions compiled with the **cc −p**.

To profile the entire program on Sun-2, Sun-3, and Sun-4 systems using the low-level interface to profil(2), it is sufficient to use

```
extern etext();
...
monitor(N_TXTOFF(0), etext, buf, bufsize, nfunc);
```

On Sun386i systems, the equivalent calls are:

```
extern etext();
extern _start();
...
monitor(_start, etext, buf, bufsize, nfunc);
```

FILES

**mon.out**

SEE ALSO

**cc**(1V), **prof**(1), **gprof**(1), **brk**(2), **profil**(2), **end**(3)

NAME

mp, itom, madd, msub, mult, mdiv, min, mout, pow, gcd, rpow, xtom, mtox, mfree – multiple precision integer arithmetic

SYNOPSIS

```
#include <mp.h>

madd(a, b, c)
MINT *a, *b, *c;

msub(a, b, c)
MINT *a, *b, *c;

mult(a, b, c)
MINT *a, *b, *c;

mdiv(a, b, q, r)
MINT *a, *b, *q, *r;

min(a)
MINT *a;

mout(a)
MINT *a;

pow(a, b, c, d)
MINT *a, *b, *c, *d;

gcd(a, b, c)
MINT *a, *b, *c;

rpow(a, n, b)
MINT *a, *b;
short n;

msqrt(a, b, r)
MINT *a, *b, *r;

sdiv(a, n, q, r)
MINT *a, *q;
short n, *r;

MINT *itom(n)
short n;

MINT *xtom(s)
char *s;

char *mtox(a)
MINT *a;

void mfree(a)
MINT *a;
```

DESCRIPTION

These routines perform arithmetic on integers of arbitrary length. The integers are stored using the defined type MINT. Pointers to a MINT should be initialized using the function *itom*, which sets the initial value to *n*. Alternatively, **xtom** may be used to initialize a MINT from a string of hexadecimal digits. **mfree()** may be used to release the storage allocated by these routines.

**madd**, **msub()** and **mult()** assign to their third arguments the sum, difference, and product, respectively, of their first two arguments. **mdiv()** assigns the quotient and remainder, respectively, to its third and fourth arguments. **sdiv** is like **mdiv()** except that the divisor is an ordinary integer. **msqrt**

produces the square root and remainder of its first argument. **mpow** calculates **a** raised to the power *b*, while **pow()** calculates this reduced modulo *m*. **min()** and **mout()** do decimal input and output. **mtox()** provides the inverse of **xtom**.

Use the **–lmp** loader option to obtain access to these functions.

**DIAGNOSTICS**

Illegal operations and running out of memory produce messages and core images.

**FILES**

**/usr/lib/libmp.a**

## NAME

ndbm, dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr – data base subroutines

## SYNOPSIS

```
#include <ndbm.h>

typedef struct {
        char *dptr;
        int dsize;
} datum;

DBM *dbm_open(file, flags, mode)
char *file;
int flags, mode;

void dbm_close (db)
DBM *db;

datum dbm_fetch(db, key)
DBM *db;
datum key;

int dbm_store(db, key, content, flags)
DBM *db;
datum key, content;
int flags;

int dbm_delete(db, key)
DBM *db;
datum key;

datum dbm_firstkey(db)
DBM *db;

datum dbm_nextkey(db)
DBM *db;

int dbm_error(db)
DBM *db;

int dbm_clearerr(db)
DBM *db;
```

## DESCRIPTION

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. This package replaces the earlier **dbm**(3X) library, which managed only a single database.

*key*s and *content*s are described by the **datum** typedef. A **datum** specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has **.dir** as its suffix. The second file contains all data and has **.pag** as its suffix.

Before a database can be accessed, it must be opened by **dbm_open**. This will open and/or create the files *file* **.dir** and *file* **.pag** depending on the flags parameter (see **open**(2V)).

A database is closed by calling **dbm_close**.

Once open, the data stored under a key is accessed by **dbm_fetch**() and data is placed under a key by **dbm_store**. The *flags* field can be either DBM_INSERT or DBM_REPLACE. DBM_INSERT will only insert new entries into the database and will not change an existing entry with the same key. DBM_REPLACE will replace an existing entry if it has the same key. A key (and its associated contents) is

contents) is deleted by **dbm_delete**. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of **dbm_firstkey()** and **dbm_nextkey**. **dbm_firstkey()** will return the first key in the database. **dbm_nextkey()** will return the next key in the database. This code will traverse the data base:

**for (key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))**

**dbm_error()** returns non-zero when an error has occurred reading or writing the database. **dbm_clearerr()** resets the error condition on the named database.

SEE ALSO

**open**(2V), **dbm**(3X)

DIAGNOSTICS

All functions that return an **int** indicate errors with negative values. A zero return indicates no error. Routines that return a **datum** indicate errors with a NULL (0) *dptr*. If **dbm_store** called with a *flags* value of **DBM_INSERT** finds an existing entry with the same key it returns 1.

BUGS

The **.pag** file will contain holes so that its apparent size is about four times its actual content. Older versions of the UNIX operating system may create real file blocks for these holes when touched. These files cannot be copied by normal means (**cp**(1), **cat**(1V), **tar**(1), **ar**(1)) without filling in the holes.

*dptr* pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 4096 bytes). Moreover all key/content pairs that hash together must fit on a single block. **dbm_store()** will return an error in the event that a disk block fills with inseparable data.

**dbm_delete()** does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by **dbm_firstkey()** and **dbm_nextkey()** depends on a hashing function, not on anything interesting.

There are no interlocks and no reliable cache flushing; thus concurrent updating and reading is risky.

**NAME**

nice – change priority of a process

**SYNOPSIS**

**int nice(incr)**

**DESCRIPTION**

The scheduling priority of the process is augmented by *incr*. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without undue impact on system performance.

Negative increments are illegal, except when specified by the super-user. The priority is limited to the range –20 (most urgent) to 20 (least). Requests for values above or below these limits result in the scheduling priority being set to the corresponding limit.

The priority of a process is passed to a child process by **fork**(2). For a privileged process to return to normal priority from an unknown state, **nice**() should be called successively with arguments –40 (goes to priority –20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

**RETURN VALUE**

Upon successful completion, **nice**() returns 0. Otherwise, a value of –1 is returned and **errno** is set to indicate the error.

**ERRORS**

The priority is not changed if:

EACCES The value of *incr* specified was negative, and the effective user ID is not super-user.

**SEE ALSO**

**nice**(1), **fork**(2), **getpriority**(2), **renice**(8)

## NAME

nlist – get entries from symbol table

## SYNOPSIS

```
#include <nlist.h>

int nlist(filename, nl)
char *filename;
struct nlist *nl;
```

## DESCRIPTION

**nlist()** examines the symbol table from the executable image whose name is pointed to by *filename*, and selectively extracts a list of values and puts them in the array of **nlist()** structures pointed to by *nl*. The name list pointed to by **nl()** consists of an array of structures containing names, types and values. The *n_name* field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a NULL pointer (or a pointer to a NULL string) in the *n_name* field. For each entry in *nl*, if the named symbol is present in the executable image's symbol table, its value and type are placed in the *n_value* and *n_type* fields. If a symbol cannot be located, the corresponding *n_type* field of **nl()** is set to zero.

## RETURN VALUE

Upon normal completion, **nlist()** returns the number of symbols that were not located in the symbol table. If an error occurs, **nlist()** returns −1 and sets all of the *n_type* fields in members of the array pointed to by **nl()** to zero.

## SEE ALSO

**a.out**(5)
**coff**(5)

## DIAGNOSTICS

On Sun-2, Sun-3, and Sun-4 systems, type entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

On Sun386i systems, the type entries may be zero even when the name list succeeded, but the value entries will be zero only when the file cannot be read or does not contain a valid name list. Therefore, on Sun386i systems, the value entry can be used to determine whether the command succeeded.

NAME

on_exit – name termination handler

SYNOPSIS

**int on_exit(procp, arg)**
**void (*procp)();**
**caddr_t arg;**

DESCRIPTION

**on_exit()** names a routine to be called after a program calls **exit**(3) or returns normally, and before its process terminates. The routine named is called as

**(*procp)(status, arg);**

where *status* is the argument with which **exit()** was called, or zero if *main* returns. Typically, *arg* is the address of an argument vector to (**procp*), but may be an integer value. Several calls may be made to **on_exit**, specifying several termination handlers. The order in which they are called is the reverse of that in which they were given to **on_exit**.

SEE ALSO

**gprof**(1), **tcov**(1), **exit**(3)

DIAGNOSTICS

**on_exit()** returns zero normally, or nonzero if the procedure name could not be stored.

BUGS

Currently there is a limit of 20 termination handlers, including any invoked implicitly (for example, by **gprof**(1) or **tcov**(1) processing). Calls to **on_exit()** beyond this number will fail.

NOTES

This call is specific to the SunOS operating system and should not be used if portability is a concern.

Standard I/O exit processing is always done last.

NAME

pause – stop until signal

SYNOPSIS

**pause()**

DESCRIPTION

**pause()** never returns normally. It is used to give up control while waiting for a signal from **kill**(2V) or an interval timer, see **getitimer**(2). Upon termination of a signal handler started during a **pause**, the **pause()** call will return.

RETURN VALUE

Always returns –1.

ERRORS

**pause()** always returns:

| | |
|---|---|
| EINTR | The call was interrupted. |

SEE ALSO

**kill**(2V), **getitimer**(2), **select**(2), **sigpause**(2)

NAME

perror, sys_errlist, sys_nerr, errno – system error messages

SYNOPSIS

```
perror(s)
char *s;
int sys_nerr;
char *sys_errlist[];
int errno;
```

DESCRIPTION

**perror()** produces a short error message on the standard error describing the last error encountered during a call to a system or library function. If *s* is not a NULL pointer and does not point to a null string, the string it points to is printed, followed by a colon, followed by a space, followed by the message and a NEWLINE. If *s* is a NULL pointer or points to a null string, just the message is printed, followed by a NEWLINE. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable **errno** (see **intro**(2)), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings **sys_errlist()** is provided; **errno** can be used as an index in this table to get the message string without the newline. **sys_nerr()** is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

**intro**(2), **psignal**(3)

NAME

plot, openpl, erase, label, line, circle, arc, move, cont, point, linemod, space, closepl – graphics interface

SYNOPSIS

**openpl()**

**erase()**

**label(s)**
**char s[];**

**line(x1, y1, x2, y2)**

**circle(x, y, r)**

**arc(x, y, x0, y0, x1, y1)**

**move(x, y)**

**cont(x, y)**

**point(x, y)**

**linemod(s)**
**char s[];**

**space(x0, y0, x1, y1)**

**closepl()**

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. See **plot**(5) for a description of their effect. **openpl()** must be used before any of the others to open the device for writing. **closepl()** flushes the output.

String arguments to **label()** and **linemod()** are NULL-terminated, and do not contain NEWLINE characters.

Various flavors of these functions exist for different output devices. They are obtained by the following **ld**(1) options:

**–lplot** device-independent graphics stream on standard output for **plot**(1G) filters

**–l300** GSI 300 terminal

**–l300s** GSI 300S terminal

**–l450** GSI 450 terminal

**–l4014** Tektronix 4014 terminal

**–lplotaed** AED 512 color graphics terminal

**–lplotbg** BBN bitgraph graphics terminal

**–lplotdumb**
Dumb terminals without cursor addressing or line printers

**–lplotgigi** DEC Gigi terminals

**–lplot2648** Hewlett Packard 2648 graphics terminal

**–lplot7221** Hewlett Packard 7221 graphics terminal

**–lplotimagen**
Imagen laser printer (default 240 dots-per-inch resolution).

FILES

**/usr/lib/libplot.a**
**/usr/lib/lib300.a**
**/usr/lib/lib300s.a**
**/usr/lib/lib450.a**
**/usr/lib/lib4014.a**
**/usr/lib/libplotaed.a**
**/usr/lib/libplotbg.a**
**/usr/lib/libplotdumb.a**
**/usr/lib/libplotgigi.a**
**/usr/lib/libplot2648.a**
**/usr/lib/libplot7221.a**
**/usr/lib/libplotimagen.a**

SEE ALSO

**graph**(1G), **ld**(1), **plot**(1G), **plot**(5)

NAME

popen, pclose – open or close a pipe (for I/O) from or to a process

SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;
```

DESCRIPTION

The arguments to **popen()** are pointers to NULL-terminated strings containing, respectively, a shell command line and an I/O mode, either **r** for reading or **w** for writing. **popen()** creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer such that one can write to the standard input of the command, if the I/O mode is **w**, by writing to the file stream; and one can read from the standard output of the command, if the I/O mode is **r**, by reading from the file stream.

A stream opened by **popen()** should be closed by **pclose**, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type **r** command may be used as an input filter, reading its standard input (which is also the standard input of the process doing the **popen**) and providing filtered input on the stream, and a type **w** command may be used as an output filter, reading a stream of output written to the stream process doing the **popen()** and further filtering it and writing it to its standard output (which is also the standard input of the process doing the **popen**).

**popen()** always calls **sh**(1), never **csh**(1).

SEE ALSO

**csh**(1), **sh**(1), **pipe**(2), **wait**(2), **fclose**(3S), **fopen**(3S), **system**(3)

DIAGNOSTICS

**popen()** returns a NULL pointer if the pipe or process cannot be created, or if it cannot allocate as much memory as it needs.

**pclose()** returns –1 if stream is not associated with a '**popened**' command.

BUGS

If the original and '**popened**' processes concurrently read or write a common file, neither should use buffered I/O, because the buffering gets all mixed up. Similar problems with an output filter may be forestalled by careful buffer flushing, for instance, with **fflush**; see **fclose**(3S).

NAME

printf, fprintf, sprintf – formatted output conversion

SYNOPSIS

```
#include <stdio.h>
int printf(format [ , arg ] ... )
char *format;

int fprintf(stream, format [ , arg ] ... )
FILE *stream;
char *format;

char *sprintf(s, format [ , arg ] ... )
char *s, *format;

#include <varargs.h>
int _doprnt(format, args, stream)
char *format;
va_list args;
FILE *stream;
```

DESCRIPTION

**printf()** places output on the standard output stream **stdout**. **fprintf()** places output on the named output **stream**. **sprintf()** places "output", followed by the NULL character (**\0**), in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough storage is available. **printf()** and **fprintf()** return the number of characters transmitted. The return value of **sprintf()** is not normally used, but cast to type **void** instead. **printf()** and **fprintf()** return an EOF if an output error was encountered.

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of zero or more *arg*s. The results are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character **%**. After the **%**, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag '–', described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

A *precision* that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, or **X** conversions, the number of digits to appear after the decimal point for the **e**, **E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string; a NULL digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional **l** (ell) specifying that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion character applies to a long integer *arg*. An **l** before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear

*before* the *arg* (if any) to be converted. A negative field width argument is taken as a '–' flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:

– The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (+ or –).

blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

# This flag specifies that the value is to be converted to an "alternate form."For **c, d, i, s,** and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x or X** conversion, a non-zero result will have **0x or 0X** prefixed to it. For **e, E, f, g,** and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

**d,i,o,u,x,X** The integer *arg* is converted to signed decimal (**d** or **i**), unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a NULL string.

**f** The float or double *arg* is converted to decimal notation in the style "[–]ddd.ddd" where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

**e,E** The float or double *arg* is converted in the style "[–]d.ddde±ddd," where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.

**g,G** The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** or **E** will be used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

The **e, E, f, g,** and **G** formats print IEEE indeterminate values (infinity or not-a-number) as "Infinity" or "NaN" respectively.

**c** The character *arg* is printed.

**s** The *arg* is taken to be a string (character pointer) and characters from the string are printed until a NULL character (\0) is encountered or until the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first NULL character are printed. A NULL value for *arg* will yield undefined results.

**%** Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Padding takes place only if the specified field width exceeds the actual width. Characters generated

by **printf()** and **fprintf()** are printed as if **putc**(3S) had been called.

EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02," where *weekday* and *month* are pointers to NULL-terminated strings:

```
printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);
```

To print $\pi$ to 5 decimal places:

```
printf("pi = %.5f", 4 * atan(1. 0));
```

NOTE

These routines call **_doprnt**, which is an implementation-dependent routine. Each uses the variable-length argument facilities of **varargs**(3). Although it is possible to use **_doprnt** to take a list of arguments and pass them on to a routine like **printf**, not all implementations have such a routine. We strongly recommend that you use the routines described in **vprintf**(3S) instead.

SEE ALSO

**econvert**(3), **printf**(3V), **putc**(3S), **scanf**(3S), **varargs**(3), **vprintf**(3S)

BUGS

Very wide fields (>128 characters) fail.

NAME

prof – profile within a function

SYNOPSIS

**#define MARK**
**#include <prof.h>**

**void MARK (name)**

DESCRIPTION

MARK will introduce a mark called *name* that will be treated the same as a function entry point. Execution of the mark will add to a counter for that mark, and program-counter time spent will be accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

*name* may be any combination of up to six letters, numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file **<prof.h>** is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, such as:

**cc –p –DMARK foo.c**

If MARK is not defined, the MARK (**name**) statements may be left in the source files containing them and will be ignored.

EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with MARK defined on the command line, the marks are ignored.

```
#include <prof.h>
func( )
{
        int i, j;
        .
        .
        .
        MARK (loop1);
        for (i = 0; i < 2000; i++) {
                . . .
        }
        MARK (loop2);
        for (j = 0; j < 2000; j++) {
                . . .
        }
}
```

SEE ALSO

**prof**(1), **profil**(2), **monitor**(3)

NAME

psignal, sys_siglist – system signal messages

SYNOPSIS

**psignal(sig, s)**
**unsigned sig;**
**char *s;**

**char *sys_siglist[ ];**

DESCRIPTION

**psignal()** produces a short message on the standard error file describing the indicated signal. First the argument string *s* is printed, then a colon, then the name of the signal and a NEWLINE. Most usefully, the argument string is the name of the program which incurred the signal. The signal number should be from among those found in **<signal.h>**.

To simplify variant formatting of signal names, the vector of message strings **sys_siglist()** is provided; the signal number can be used as an index in this table to get the signal name without the newline. The define NSIG defined in **<signal.h>** is the number of messages provided for in the table; it should be checked because new signals may be added to the system before they are added to the table.

SEE ALSO

**perror**(3), **signal**(3)

NAME

putc, putchar, fputc, putw – put character or word on a stream

SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

int putchar(c)
char c;

int fputc(c, stream)
char c;
FILE *stream;

int putw(w, stream)
int w;
FILE *stream;
```

DESCRIPTION

**putc**( ) writes the character *c* onto the standard I/O output stream *stream* (at the position where the file pointer, if defined, is pointing). It returns the character written.

**putchar**(c) is defined as **putc**(c, **stdout**). **putc**( ) and **putchar**( ) are macros.

**fputc**( ) behaves like **putc**, but is a function rather than a macro. **fputc**( ) runs more slowly than **putc**, but it takes less space per invocation and its name can be passed as an argument to a function.

**putw**( ) writes the C **int** (word) **w** to the standard I/O output stream *stream* (at the position of the file pointer, if defined). The size of a word is the size of an integer and varies from machine to machine. **putw**( ) neither assumes nor causes special alignment in the file.

Output streams are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. When an output stream is unbuffered, information is queued for writing on the destination file or terminal as soon as written; when it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (that is, as soon as a NEWLINE character is written or terminal input is requested). **setbuf**(3S), **setbuffer**, or **setvbuf** may be used to change the stream's buffering strategy.

SEE ALSO

**fclose**(3S), **ferror**(3S), **fopen**(3S), **fread**(3S), **getc**(3S), **printf**(3S), **puts**(3S), **setbuf**(3S),

DIAGNOSTICS

On success, **putc**, **fputc**, and **putchar** return the value that was written. On error, those functions return the constant **EOF**. **putw**( ) returns **ferror**(**stream**), so that it returns 0 on success and 1 on failure.

BUGS

Because it is implemented as a macro, **putc**( ) treats a *stream* argument with side effects improperly. In particular, **putc**(c, ***f++**); does not work sensibly. **fputc**( ) should be used instead.

Errors can occur long after the call to **putc**.

Because of possible differences in word length and byte ordering, files written using **putw**( ) are machine-dependent, and may not be read using **getw**( ) on a different processor.

NAME

putenv – change or add value to environment

SYNOPSIS

**int putenv(string)**
**char *string;**

DESCRIPTION

**string()** points to a string of the form '*name* = *value*' **putenv()** makes the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by **string()** becomes part of the environment, so altering the string will change the environment. The space used by **string()** is no longer used once a new string-defining *name* is passed to **putenv**.

SEE ALSO

**execve**(2), **getenv**(3), **malloc**(3), **environ**(5V).

DIAGNOSTICS

**putenv()** returns non-zero if it was unable to obtain enough space using **malloc**(3) for an expanded environment, otherwise zero.

WARNINGS

**putenv()** manipulates the environment pointed to by *environ*, and can be used in conjunction with **getenv**. However, *envp* (the third argument to *main*) is not changed.

This routine uses **malloc**(3) to enlarge the environment.

After **putenv()** is called, environmental variables are not in alphabetical order.

A potential error is to call **putenv()** with an automatic variable as the argument, then exit the calling function while **string()** is still part of the environment.

NAME

putpwent – write password file entry

SYNOPSIS

```
#include <pwd.h>

int putpwent(p, f)
struct passwd *p;
FILE *f;
```

DESCRIPTION

**putpwent()** is the inverse of **getpwent**(3). Given a pointer to a passwd structure created by **getpwent()** (or **getpwuid()** or **getpwnam**), **putpwent()** writes a line on the stream *f*, which matches the format of lines in the password file **/etc/passwd**.

FILES

**/etc/passwd**

SEE ALSO

**getpwent**(3)

DIAGNOSTICS

**putpwent()** returns non-zero if an error was detected during its operation, otherwise zero.

WARNING

The above routine uses **<stdio.h>**, which increases the size of programs, not otherwise using standard I/O, more than might be expected.

BUGS

This routine is of limited utility, since most password files are maintained as Yellow Pages files, and cannot be updated with this routine.

NAME

puts, fputs – put a string on a stream

SYNOPSIS

```
#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

**puts()** writes the NULL-terminated string pointed to by *s*, followed by a NEWLINE character, to the standard output stream **stdout**.

**fputs()** writes the NULL-terminated string pointed to by *s* to the named output stream.

Neither function writes the terminal SM NULL character.

DIAGNOSTICS

Both routines return EOF on error. This will happen if the routines try to write on a file that has not been opened for writing.

NOTES

**puts()** appends a NEWLINE while **fputs()** does not.

SEE ALSO

**ferror**(3S), **fopen**(3S), **fread**(3S), **printf**(3S), **putc**(3S)

## NAME

pwdauth, grpauth – password authentication routines

## SYNOPSIS

**int pwdauth(user, password)**
**char *user;**
**char *password;**

**int grpauth(group, password)**
**char *group;**
**char *password;**

## DESCRIPTION

**pwdauth()** and **grpauth()** determine whether the given guess at a *password* is valid for the given *user* or *group*. If the *password* is valid, the functions return 0.

A *password* is valid if the password when encrypted matches the encrypted password in the appropriate file. For **pwdauth**, if the **password.adjunct** file exists, the encrypted password will be in either the local or the Yellow Pages version of that file. Otherwise, either the local or YP **passwd** file will be used. For **grpauth**, the **group.adjunct** file (if it exists) or the **group** file (otherwise) will be checked on the local machine and then using the YP. In all cases, the local files will be checked before the YP files. Also, if the adjunct files exist, the main file will never be used for authentication even if they include encrypted passwords.

Both **pwdauth()** and **grpauth()** interface to the authentication daemon, **rpc.pwdauthd**, to do the checking of the adjunct files. This daemon must be running on any system that provides password authentication.

## FILES

**/etc/passwd**
**/etc/group**

## SEE ALSO

**getgraent**(3), **getgrent**(3), **getpwaent**(3), **getpwent**(3), **pwdauthd**(8C)

NAME

qsort – quicker sort

SYNOPSIS

```
qsort(base, nel, width, compar)
char *base;
int (*compar)();
```

DESCRIPTION

**qsort()** is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

*base* points to the element at the base of the table. *nel* is the number of elements in the table. *width* is the size, in bytes, of each element in the table. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. As the function must return an integer less than, equal to, or greater than zero, so must the first argument to be considered be less than, equal to, or greater than the second.

NOTES

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The order in the output of two items which compare as equal is unpredictable.

SEE ALSO

**sort**(1V), **bsearch**(3), **lsearch**(3), **string**(3)

EXAMPLE

The following program sorts a simple array:

```
static  int intcompare(i,j)
        int *i, *j;
{
        return(*i – *j);
}

main()
{
        int a[10];
        int i;

        a[0] = 9;
        a[1] = 8;
        a[2] = 7;
        a[3] = 6;
        a[4] = 5;
        a[5] = 4;
        a[6] = 3;
        a[7] = 2;
        a[8] = 1;
        a[9] = 0;

        qsort(a,10,sizeof(int),intcompare)

        for (i=0; i<10; i++) printf(" %d,a[i]);
        printf "0);
}
```

NAME

rand, srand – simple random number generator

SYNOPSIS

**srand(seed)**
**int seed;**

**rand()**

DESCRIPTION

**rand()** uses a multiplicative congruential random number generator with period $2^{32}$ to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$.

**srand()** can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NOTE

The spectral properties of **rand()** leave a great deal to be desired. **drand48**(3) and **random**(3) provide much better, though more elaborate, random-number generators.

SEE ALSO

**drand48**(3), **random**(3), **rand**(3V)

BUGS

The low bits of the numbers generated are not very random; use the middle bits. In particular the lowest bit alternates between 0 and 1.

NAME

random, srandom, initstate, setstate – better random number generator; routines for changing generators

SYNOPSIS

```
long random()

srandom(seed)
int seed;

char *initstate(seed, state, n)
unsigned seed;
char *state;
int n;

char *setstate(state)
char *state;
```

DESCRIPTION

**random()** uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random number generator is very large, approximately $16\times(2^{31}-1)$.

**random/srandom** have (almost) the same calling sequence and initialization properties as **rand/srand**. The difference is that **rand**(3C) produces a much less random sequence — in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by **random()** are usable. For example,

```
random()&01
```

will produce a random binary value.

Unlike **srand, srandom()** does not return the old seed; the reason for this is that the amount of state information used is much more than a single word. (Two other routines are provided to deal with restarting/changing random number generators). Like **rand**(3C), however, **random()** will by default produce a sequence of numbers that can be duplicated by calling **srandom()** with *1* as the seed.

The **initstate()** routine allows a state array, passed in as an argument, to be initialized for future use. The size of the state array (in bytes) is used by **initstate()** to decide how sophisticated a random number generator it should use — the more state, the better the random numbers will be. (Current "optimal" values for the amount of state information are 8, 32, 64, 128, and 256 bytes; other amounts will be rounded down to the nearest known amount. Using less than 8 bytes will cause an error). The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. **initstate()** returns a pointer to the previous state information array.

Once a state has been initialized, the **setstate()** routine provides for rapid switching between states. **setstate()** returns a pointer to the previous state array; its argument state array is used for further random number generation until the next call to **initstate()** or **setstate**.

Once a state array has been initialized, it may be restarted at a different point either by calling **initstate()** (with the desired seed, the state array, and its size) or by calling both **setstate()** (with the state array) and **srandom()** (with the desired seed). The advantage of calling both **setstate()** and **srandom()** is that the size of the state array does not have to be remembered after it is initialized.

With 256 bytes of state information, the period of the random number generator is greater than $2^{69}$, which should be sufficient for most purposes.

SEE ALSO

**rand**(3C)

EXAMPLE

```
/* Initialize and array and pass it in to initstate. */

static long state1[32] = {
        3,
        0x9a319039, 0x32d9c024, 0x9b663182, 0x5da1f342,
        0x7449e56b, 0xbeb1dbb0, 0xab5c5918, 0x946554fd,
        0x8c2e680f, 0xeb3d799f, 0xb11ee0b7, 0x2d436b86,
        0xda672e2a, 0x1588ca88, 0xe369735d, 0x904f35f7,
        0xd7158fd6, 0x6fa6f051, 0x616e6b96, 0xac94efdc,
        0xde3b81e0, 0xdf0a6fb5, 0xf103bc02, 0x48f340fb,
        0x36413f93, 0xc622c298, 0xf5a42ab8, 0x8a88d77b,
        0xf5ad9d0e, 0x8999220b, 0x27fb47b9
        };

main()
{
        unsigned seed;
        int n;

        seed = 1;
        n = 128;
        initstate(seed, state1, n);

        setstate(state1);
        printf("%d0,random());
}
```

DIAGNOSTICS

If **initstate( )** is called with less than 8 bytes of state information, or if **setstate( )** detects that the state information has been garbled, error messages are printed on the standard error output.

BUGS

About 2/3 the speed of **rand**(3C).

NAME

rcmd, rresvport, ruserok – routines for returning a stream to a remote command

SYNOPSIS

**int rcmd(ahost, inport, locuser, remuser, cmd, fd2p)**
**char **ahost;**
**int inport;**
**char *locuser, *remuser, *cmd;**
**int *fd2p**

**int rresvport(port)**
**int *port;**

**ruserok(rhost, super-user, ruser, luser)**
**char *rhost;**
**int super-user;**
**char *ruser, *luser;**

DESCRIPTION

**rcmd()** is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. **rresvport()** is a routine which returns a descriptor to a socket with an address in the privileged port space. **ruserok()** is a routine used by servers to authenticate clients requesting service with **rcmd**. All three functions are present in the same file and are used by the **rshd**(8C) server (among others).

**rcmd()** looks up the host **ahost* using **gethostbyname** (see **gethostent**(3N)), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port *inport*.

If the connection succeeds, a socket in the Internet domain of type **SOCK_STREAM** is returned to the caller, and given to the remote command as its standard input (file descriptor 0) and standard output (file descriptor 1). If *fd2p* is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (file descriptor 2) on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the standard error (file descriptor 2) of the remote command will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in **rshd**(8C).

The **rresvport()** routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by **rcmd()** and several other routines. Privileged Internet ports are those in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to a socket.

**ruserok()** takes a remote host's name, as returned by a **gethostbyaddr** (see **gethostent**(3N)) routine, two user names and a flag indicating whether the local user's name is that of the super-user. It then checks the files **/etc/hosts.equiv** and, possibly, **.rhosts** in the local user's home directory to see if the request for service is allowed. A 0 is returned if the machine name is listed in the **/etc/hosts.equiv** file, or the host and remote user name are found in the **.rhosts** file; otherwise **ruserok()** returns –1. If the super-user flag is 1, the checking of the **/etc/hosts.equiv** file is bypassed.

FILES

**/etc/hosts.equiv**
**.rhosts**

SEE ALSO

**rlogin**(1C), **rsh**(1C), **intro**(2), **gethostent**(3N), **rexec**(3N), **rexecd**(8C), **rlogind**(8C), **rshd**(8C)

DIAGNOSTICS

**rcmd()** returns a valid socket descriptor on success. It returns −1 on error and prints a diagnostic message on the standard error.

**rresvport()** returns a valid, bound socket descriptor on success. It returns −1 on error with the global value **errno** set according to the reason for failure. The error code EAGAIN is overloaded to mean "All network ports in use."

NAME

realpath – returns the real file name.

SYNOPSIS

**char *realpath(file_name, resolved_name)**
**char *file_name, resolved_name;**

DESCRIPTION

**realpath()** resolves all links and all references to "." and ".." in *file_name* and stores it in *resolved_name*.

It can handle both relative and absolute path names.

RETURN VALUE

If there is no error, it returns a pointer to the *resolved_name*. Otherwise it returns a NULL pointer and places the name of the offending file in *resolved_name*.

SEE ALSO

**getwd(3)**

WARNINGS

It operates on null-terminated strings.

It does not check for overflow of the receiving string.

NAME

regex, re_comp, re_exec – regular expression handler

SYNOPSIS

**char *re_comp(s)**
**char *s;**

**re_exec(s)**
**char *s;**

DESCRIPTION

**re_comp()** compiles a string into an internal form suitable for pattern matching. **re_exec()** checks the argument string against the last string passed to **re_comp**.

**re_comp()** returns a NULL pointer if the string *s* was compiled successfully; otherwise a string containing an error message is returned. If **re_comp()** is passed 0 or a NULL string, it returns without changing the currently compiled regular expression.

**re_exec()** returns 1 if the string *s* matches the last compiled regular expression, 0 if the string *s* failed to match the last compiled regular expression, and −1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both **re_comp()** and **re_exec()** may have trailing or embedded NEWLINE characters; they are terminated by NULL characters. The regular expressions recognized are described in the manual entry for **ed**(1), given the above difference.

SEE ALSO

**ed**(1), **ex**(1), **grep**(1V)

DIAGNOSTICS

**re_exec()** returns −1 for an internal error.

**re_comp()** returns one of the following strings if an error occurs:

**No previous regular expression**

**Regular expression too long**

**unmatched \(**

**missing ]**

**too many \(\) pairs**

**unmatched \)**

## NAME

regexp – regular expression compile and match routines

## SYNOPSIS

```
#define INIT <declarations>
#define GETC( ) <getc code>
#define PEEKC( ) <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step(string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;
```

## DESCRIPTION

This page describes general-purpose regular expression matching routines.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the '**#include <regexp.h>**' statement. These macros are used by the *compile* routine.

**GETC( )** Return the value of the next character in the regular expression pattern. Successive calls to GETC( ) should return successive characters of the regular expression.

**PEEKC( )** Return the next character in the regular expression. Successive calls to PEEKC( ) should return the same character, which should also be the next character returned by GETC( ).

**UNGETC(*c*)** Returns the argument *c* by the next call to GETC( ) or PEEKC( ). No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC( ). The value of the macro UNGETC(*c*) is always ignored.

**RETURN(*pointer*)** This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs that have memory allocation to manage.

## ERRORS

**ERROR(*val*)** This is the abnormal return from the **compile( )** routine. The argument *val* is an error number (see table below for meanings). This call should never return.

| ERROR | MEANING |
|---|---|
| 11 | Range endpoint too large. |
| 16 | Bad number. |
| 25 | "\ digit" out of range. |
| 36 | Illegal or missing delimiter. |
| 41 | No remembered search string. |
| 42 | \( \) imbalance. |
| 43 | Too many \(. |

| | |
|---|---|
| 44 | More than 2 numbers given in \{ \}. |
| 45 | } expected after \. |
| 46 | First number exceeds second in \{ \}. |
| 49 | [ ] imbalance. |
| 50 | Regular expression too long. |

The syntax of the **compile()** routine is as follows:

**compile(instring, expbuf, endbuf, eof)**

The first parameter *instring* is never used explicitly by the **compile()** routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the **INIT()** declaration (see below). Programs that call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf–expbuf*) bytes, a call to **ERROR(50)** is made.

The parameter *eof* is the character that marks the end of the regular expression. For example, in an editor like **ed**(1), this character would usually a '/'.

Each program that includes this file must have a **#define** statement for **INIT()**. This definition will be placed right after the declaration for the function **compile()** and '{' (opening curly brace). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for **GETC()**, **PEEKC()**, and **UNGETC()**. Otherwise it can be used to declare external variables that might be used by **GETC()**, **PEEKC()**, and **UNGETC()**. See the example below of the declarations taken from **grep**(1V).

There are other functions in this file that perform actual regular expression matching, one of which is the function **step()**. The call to **step()** is as follows:

**step(string, expbuf)**

The first parameter to **step()** is a pointer to a string of characters to be checked for a match. This string should be NULL-terminated.

The second parameter *expbuf* is the compiled regular expression that was obtained by a call of the function *compile*.

The function **step()** returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to **step()**. The variable set in **step()** is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function **advance()**, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, **loc1** will point to the first character of *string* and **loc2** will point to the NULL at the end of *string*.

**step()** uses the external variable **circf** which is set by **compile()** if the regular expression begins with '^'. If this is set then **step()** will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed the value of **circf** should be saved for each compiled expression and **circf** should be set to that saved value before each call to **step()**.

The function **advance()** is called from **step()** with the same arguments as **step()**. The purpose of **step()** is to step through the *string* argument and call **advance()** until **advance()** returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, **step()** need not be called; simply call **advance()**.

When **advance()** encounters a * or \{ \} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, **advance()** will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{ \}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer **locs** is equal to the point in the string at sometime during the backing up process, **advance()** will break out of the loop that backs up and will return zero. This could be used by an editor like **ed**(1) or **sed**(1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like **s/y*//g** do not loop forever.

The additional external variables **sed** and **nbra** are used for special purposes.

## EXAMPLES

The following is an example of how the regular expression macros and calls could look in a command like **grep**(1V):

```
#define INIT    register char *sp = instring;
#define GETC() (*sp++)
#define PEEKC()         (*sp)
#define UNGETC(c)       (--sp)
#define RETURN(c)       return;
#define ERROR(c)        regerr()

#include <regexp.h>
...
                (void) compile(*argv, expbuf, &expbuf[ESIZE], '\0');
...
                if (step(linebuf, expbuf))
                        succeed ();
```

## FILES

**/usr/include/regexp.h**

## SEE ALSO

**ed**(1), **grep**(1V), **sed**(1V)

## BUGS

The handling of **circf** is difficult.

NAME

resolver, res_mkquery, res_send, res_init, dn_comp, dn_expand – resolver routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <arpa/resolv.h>

res_mkquery(op, dname, class, type, data, datalen, newrr, buf, buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init()

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
char *exp_dn, *comp_dn;
int length;
char **dnptrs, **lastdnptr;

dn_expand(msg, msglen, comp_dn, exp_dn, length)
char *msg, *comp_dn, exp_dn;
int msglen, length;
```

DESCRIPTION

These routines are used for making, sending and interpreting packets to Internet domain name servers. Global information that is used by the resolver routines is kept in the variable *_res*. Most of the values have reasonable defaults and can be ignored. Options stored in *_res.options* are defined in *resolv.h* and are as follows. Options are a simple bit mask and are OR'ed in to enable.

| | |
|---|---|
| **RES_INIT** | True if the initial name server address and default domain name are initialized (that is, *res_init* has been called). |
| **RES_DEBUG** | Print debugging messages. |
| **RES_AAONLY** | Accept authoritative answers only. *res_send* will continue until it finds an authoritative answer or finds an error. Currently this is not implemented. |
| **RES_USEVC** | Use TCP connections for queries instead of UDP. |
| **RES_STAYOPEN** | Used with **RES_USEVC** to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used. |
| **RES_IGNTC** | Unused currently (ignore truncation errors, that is, do not retry with TCP). |
| **RES_RECURSE** | Set the recursion desired bit in queries. This is the default. (*res_send* does not do iterative queries and expects the name server to handle recursion.) |
| **RES_DEFNAMES** | Append the default domain name to single label queries. This is the default. |

*res_init* reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried. *res_mkquery* makes a standard query message and places it in *buf*. *res_mkquery* will return the size of the query or −1 if the query is larger than *buflen*. *op* is usually **QUERY** but can be any of the query types defined in *nameser.h*. *dname* is the domain name. If *dname* consists of a single label and the **RES_DEFNAMES** flag is enabled (the default), *dname* will be appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable **LOCALDOMAIN**. *newrr* is currently unused but is intended for making update messages.

*res_send* sends a query to name servers and returns an answer. It will call *res_init* if **RES_INIT** is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned or −1 if there were errors.

*dn_expand* Expands the compressed domain name *comp_dn* to a full domain name. Expanded names are converted to upper case. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or −1 if there was an error.

*dn_comp* Compresses the domain name *exp_dn* and stores it in *comp_dn*. The size of the compressed name is returned or −1 if there were errors. *length* is the size of the array pointed to by *comp_dn*. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to to the beginning of the message and the list ends with NULL. *lastdnptr* is a pointer to the end of the array pointed to *dnptrs*. A side effect is to update the list of pointers for labels inserted into the message by *dn_comp* as the name is compressed. If *dnptr* is NULL, we do not try to compress names. If *lastdnptr* is NULL, we do not update the list.

FILES

**/etc/resolve.conf** see **resolve.conf(5)**

SEE ALSO

**resolve.conf**(5), **named**(8)

## NAME

rexec – return stream to a remote command

## SYNOPSIS

```
rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
u_short inport;
char *user, *passwd, *cmd;
int *fd2p;
```

## DESCRIPTION

**rexec( )** looks up the host **ahost* using **gethostbyname** (see **gethostent**(3N)), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's **.netrc** file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port **inport** specifies which well-known DARPA Internet port to use for the connection; it will normally be the value returned from the call **getservbyname("exec", "tcp")** (see **getservent**(3N)). The protocol for connection is described in detail in **rexecd**(8C).

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as its standard input and standard output. If *fd2p* is non-zero, then a auxiliary channel to a control process will be setup, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the standard error (unit 2 of the remote command) will be made the same as its standard output and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

## SEE ALSO

**gethostent**(3N), **getservent**(3N), **rcmd**(3N), **rexecd**(8C)

## BUGS

There is no way to specify options to the **socket( )** call that **rexec( )** makes.

NAME

rpc – library routines for remote procedure calls

SYNOPSIS AND DESCRIPTION

These routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a data packet to the server. Upon receipt of the packet, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

**#include <rpc/rpc.h>**

**void**
**auth_destroy(auth)**
**AUTH *auth;**

A macro that destroys the authentication information associated with *auth*. Destruction usually involves deallocation of private data structures. The use of *auth* is undefined after calling **auth_destroy()**.

**AUTH ***
**authnone_create()**

Create and returns an RPC authentication handle that passes nonusable authentication information with each remote procedure call. This is the default authentication used by RPC.

**AUTH ***
**authdes_create(name, window, syncaddr, ckey)**
**char *name;**
**unsigned window;**
**struct sockaddr *addr;**
**des_block *ckey;**

**authdes_create()** is the first of two routines which interface to the RPC secure authentication system, known as DES authentication. The second is **authdes_getucred()**, below. Note: the keyserver daemon **keyserv**(8C) must be running for the DES authentication system to work.

**authdes_create()**, used on the client side, returns an authentication handle that will enable the use of the secure authentication system. The first parameter *name* is the network name, or *netname*, of the owner of the server process. This field usually represents a *hostname* derived from the utility routine **host2netname**, but could also represent a user name using **user2netname**. The second field is window on the validity of the client credential, given in seconds. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift. The third parameter *syncaddr* is optional. If it is NULL, then the authentication system will assume that the local clock is always in sync with the server's clock, and will not attempt resynchronizations. If an address is supplied, however, then the system will use the address for consulting the remote time service whenever resynchronization is required. This parameter is usually the address of the RPC server itself. The final parameter *ckey* is also optional. If it is NULL, then the authentication system will generate a random DES key to be used for the encryption of credentials. If it is supplied, however, then it will be used instead.

**authdes_getucred(adc, uid, gid, grouplen, groups)**
**struct authdes_cred *adc;**
**short *uid;**
**short *gid;**
**short *grouplen;**
**int *groups;**

**authdes_getucred()**, the second of the two DES authentication routines, is used on the server side for converting a DES credential, which is operating system independent, into a credential. This routine differs from utility routine **netname2user** in that **authdes_getucred()** pulls its information from a cache, and does not have to do a Yellow Pages lookup every time it is called to get its information.

**AUTH ***
**authunix_create(host, uid, gid, len, aup_gids)**
**char *host;**
**int uid, gid, len, *aup.gids;**

Create and return an RPC authentication handle that contains authentication information. The parameter *host* is the name of the machine on which the information was created; *uid* is the user's user ID ; *gid* is the user's current group ID ; *len* and *aup_gids* refer to a counted array of groups to which the user belongs. It is easy to impersonate a user.

**AUTH ***
**authunix_create_default()**

Calls **authunix_create()** with the appropriate parameters.

**callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)**
**char *host;**
**u_long prognum, versnum, procnum;**
**char *in, *out;**
**xdrproc_t inproc, outproc;**

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s); *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results. This routine returns zero if it succeeds, or the value of **enum clnt_stat** cast to an integer if it fails. The routine **clnt_perrno()** is handy for translating failure statuses into messages.

Warning: calling remote procedures with this routine uses UDP/IP as a transport; see **clntudp_create()** for restrictions. You do not have control of timeouts or authentication using this routine.

**enum clnt_stat**
**clnt_broadcast(prognum, versnum, procnum, inproc, in, outproc, out, eachresult)**
**u_long prognum, versnum, procnum;**
**char *in, *out;**
**xdrproc_t inproc, outproc;**
**resultproc_t eachresult;**

Like **callrpc()**, except the call message is broadcast to all locally connected broadcast nets. Each time it receives a response, this routine calls **eachresult()**, whose form is:

**eachresult(out, addr)**
**char *out;**
**struct sockaddr_in *addr;**

where *out* is the same as *out* passed to **clnt_broadcast()**, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results. If **eachresult()** returns zero, **clnt_broadcast()** waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast sockets are limited in size to the maximum transfer unit of the data link. For ethernet, this value is 1500 bytes.

**enum clnt_stat**
**clnt_call(clnt, procnum, inproc, in, outproc, out, tout)**
**CLIENT *clnt;**
**u_long**
**procnum;**
**xdrproc_t inproc, outproc;**
**char *in, *out;**
**struct timeval tout;**

A macro that calls the remote procedure *procnum* associated with the client handle, *clnt*, which is obtained with an RPC client creation routine such as **clnt_create()**. The parameter *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s); *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *tout* is the time allowed for results to come back.

**clnt_destroy(clnt)**
**CLIENT *clnt;**

A macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling **clnt_destroy()**. If the RPC library opened the associated socket, it will close it also. Otherwise, the socket remains open.

**CLIENT ***
**clnt_create(host, prog, vers, proto)**
**char *host;**
**u_long prog, vers;**
**char *proto;**

Generic client creation routine. *host* identifies the name of the remote host where the server is located. *proto* indicates which kind of transport protocol to use. The currently supported values for this field are "udp" and "tcp". Default timeouts are set, but can be modified using **clnt_control()**.

Warning: Using UDP has its shortcomings. Since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

**bool_t**
**clnt_control(cl, req, info)**
**CLIENT *cl;**
**char *info;**

A macro used to change or retrieve various information about a client object. *req* indicates the type of operation, and *info* is a pointer to the information. For both UDP and TCP, the supported values of *req* and their argument types and what they do are:

| | | |
|---|---|---|
| CLSET_TIMEOUT | struct timeval | set total timeout |
| CLGET_TIMEOUT | struct timeval | get total timeout |

Note: if you set the timeout using **clnt_control()**, the timeout parameter passed to **clnt_call()** will be ignored in all future calls.

| | | |
|---|---|---|
| CLGET_SERVER_ADDR | struct sockaddr | get server's address |

The following operations are valid for UDP only:

| | |
|---|---|
| CLSET_RETRY_TIMEOUT | struct timevalset the retry timeout |
| CLGET_RETRY_TIMEOUT | struct timevalget the retry timeout |

The retry timeout is the time that UDP RPC waits for the server to reply before retransmitting the request.

**clnt_freeres(clnt, outproc, out)**
**CLIENT *clnt;**
**xdrproc_t outproc;**
**char *out;**

A macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter *out* is the address of the results, and *outproc* is the XDR routine describing the results. This routine returns one if the results were successfully freed, and zero otherwise.

**void**
**clnt_geterr(clnt, errp)**
**CLIENT *clnt;**
**struct rpc_err *errp;**

A macro that copies the error structure out of the client handle to the structure at address *errp*.

**void**
**clnt_pcreateerror(s)**
**char *s;**

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with string *s* and a colon. Used when a **clnt_create()**, **clntraw_create()**, **clnttcp_create()**, or **clntudp_create()** call fails.

**void**
**clnt_perrno(stat)**
**enum clnt_stat stat;**

Print a message to standard error corresponding to the condition indicated by *stat*. Used after **callrpc()**.

**clnt_perror(clnt, s)**
**CLIENT *clnt;**
**char *s;**

Print a message to standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. Used after **clnt_call()**.

**char ***
**clnt_spcreateerror**
**char *s;**

Like **clnt_pcreateerror()**, except that it returns a string instead of printing to the standard error.

Bugs: returns pointer to static data that is overwritten on each call.

**char ***
**clnt_sperrno(stat)**
**enum clnt_stat stat;**

Take the same arguments as **clnt_perrno()**, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message. The string ends with a NEWLINE.

**clnt_sperrno()** is used instead of **clnt_perrno()** if the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with **printf**, or if a message format different than that supported by **clnt_perrno()** is to be used. Note: unlike **clnt_sperror()** and **clnt_spcreaterror()**, **clnt_sperrno()** does not return pointer to static data so the result will not get overwritten on each call.

**char ***
**clnt_sperror(rpch, s)**
**CLIENT *rpch;**
**char *s;**

Like **clnt_perror()**, except that (like **clnt_sperrno()**) it returns a string instead of printing to standard error.

Bugs: returns pointer to static data that is overwritten on each call.

**CLIENT ***
**clntraw_create(prognum, versnum)**
**u_long prognum, versnum;**

This routine creates a toy RPC client for the remote program *prognum*, version *versnum*. The transport used to pass messages to the service is actually a buffer within the process's address space, so the corresponding RPC server should live in the same address space; see **svcraw_create()**. This allows simulation of RPC and acquisition of RPC overheads, such as round trip times, without any kernel interference. This routine returns NULL if it fails.

**CLIENT ***
**clnttcp_create(addr, prognum, versnum, sockp, sendsz, recvsz)**
**struct sockaddr_in *addr;**
**u_long prognum, versnum;**
**int *sockp;**
**u_int sendsz, recvsz;**

This routine creates an RPC client for the remote program *prognum*, version *versnum*; the client uses TCP/IP as a transport. The remote program is located at Internet address **addr*. If **addr->sin_port** is zero, then it is set to the actual port that the remote program is listening on (the remote **portmap** service is consulted for this information). The parameter *sockp* is a socket; if it is **RPC_ANYSOCK**, then this routine opens a new one and sets *sockp*. Since TCP-based RPC uses buffered I/O , the user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of zero choose suitable defaults. This routine returns NULL if it fails.

**CLIENT ***
**clntudp_create(addr, pronum, versnum, wait, sockp)**
**struct sockaddr_in *addr;**
**u_long prognum, versnum;**
**struct timeval wait;**
**int *sockp;**

This routine creates an RPC client for the remote program *prognum*, version *versnum*; the client uses use UDP/IP as a transport. The remote program is located at Internet address *addr*. If **addr->sin_port** is zero, then it is set to actual port that the remote program is listening on (the remote **portmap** service is consulted for this information). The parameter *sockp* is a socket; if it is **RPC_ANYSOCK**, then this routine opens a new one and sets *sockp*. The UDP transport resends the call message in intervals of **wait** time until a response is received or until the call times out. The total time for the call to time out is specified by **clnt_call()**.

Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

**host2netname(name, host, domain)**
**char *name;**
**char *host;**
**char *domain;**

Convert from a domain-specific hostname to an operating-system independent netname. Return TRUE if it succeeds and FALSE if it fails. Inverse of **netname2host()**.

**key_decryptsession(remotename, deskey)**
**char *remotename;**
**des_block *deskey;**

**key_decryptsession()** is an interface to the keyserver daemon, which is associated with RPC's secure authentication system (DES authentication). User programs rarely need to call it, or its associated routines **key_encryptsession()**, **key_gendes()** and **key_setsecret()**. System commands such as **login** and the RPC library are the main clients of these four routines.

**key_decryptsession()** takes a server netname and a des key, and decrypts the key by using the the public key of the the server and the secret key associated with the effective uid of the calling process. It is the inverse of **key_encryptsession()**.

**key_encryptsession(remotename, deskey)**
**char *remotename;**
**des_block *deskey;**

**key_encryptsession()** is a keyserver interface routine. It takes a server netname and a des key, and encrypts it using the public key of the the server and the secret key associated with the effective uid of the calling process. It is the inverse of **key_decryptsession()**.

**key_gendes(deskey)**
**des_block *deskey;**

**key_gendes()** is a keyserver interface routine. It is used to ask the keyserver for a secure conversation key. Choosing one at "random" is usually not good enough, because the common ways of choosing random numbers, such as using the current time, are very easy to guess.

**key_setsecret(key)**
**char *key;**

**key_setsecret()** is a keyserver interface routine. It is used to set the key for the effective *uid* of the calling process.

**void**
**get_myaddress(addr)**
**struct sockaddr_in *addr;**

Stuff the machine's IP address into **addr*, without consulting the library routines that deal with **/etc/hosts**. The port number is always set to **htons(PMAPPORT)**.

**getnetname(name)**
**char name[MAXNETNAMELEN];**

**getnetname()** installs the unique, operating-system independent netname of the caller in the fixed-length array *name*. Returns TRUE if it succeeds and FALSE if it fails.

**netname2host(name, host, hostlen)**
**char *name;**
**char *host;**
**int hostlen;**

Convert from an operating-system independent netname to a domain-specific hostname. Returns TRUE if it succeeds and FALSE if it fails. Inverse of **host2netname()**.

**netname2user(name, uidp, gidp, gidlenp, gidlist)**
**char *name;**
**int *uidp;**
**int *gidp;**
**int *gidlenp;**
**int *gidlist;**

Convert from an operating-system independent netname to a domain-specific user ID. Returns TRUE if it succeeds and FALSE if it fails. Inverse of **user2netname()**.

**struct pmaplist ***
**pmap_getmaps(addr)**
**struct sockaddr_in *addr;**

A user interface to the **portmap** service, which returns a list of the current RPC program-to-port mappings on the host located at IP address **addr*. This routine can return NULL . The command '**rpcinfo** –**p**' uses this routine.

**u_short**
**pmap_getport(addr, prognum, versnum, protocol)**
**struct sockaddr_in *addr;**
**u_long prognum, versnum, protocol;**

A user interface to the **portmap** service, which returns the port number on which waits a service that supports program number *prognum*, version *versnum*, and speaks the transport protocol associated with *protocol*. The value of *protocol* is most likely **IPPROTO_UDP** or **IPPROTO_TCP**. A return value of zero means that the mapping does not exist or that the RPC system failured to contact the remote **portmap** service. In the latter case, the global variable **rpc_createerr()** contains the RPC status.

**enum clnt_stat**
**pmap_rmtcall(addr, prognum, versnum, procnum, inproc, in, outproc, out, tout, portp)**
**struct sockaddr_in *addr;**
**u_long prognum, versnum, procnum;**
**char *in, *out;**
**xdrproc_t inproc, outproc;**
**struct timeval tout;**
**u_long *portp;**

A user interface to the **portmap** service, which instructs **portmap** on the host at IP address **addr* to make an RPC call on your behalf to a procedure on that host. The parameter **portp* will be modified to the program's port number if the procedure succeeds. The definitions of other parameters are discussed in **callrpc()** and **clnt_call()**. This procedure should be used for a "ping" and nothing else. See also **clnt_broadcast()**.

**pmap_set(prognum, versnum, protocol, port)**
**u_long prognum, versnum, protocol;**
**u_short port;**

A user interface to the **portmap** service, which establishes a mapping between the triple [*prognum*,*versnum*,*protocol*] and *port* on the machine's **portmap** service. The value of *protocol* is most likely IPPROTO_UDP or IPPROTO_TCP. This routine returns one if it succeeds, zero otherwise. Automatically done by **svc_register()**.

**pmap_unset(prognum, versnum)**
**u_long prognum, versnum;**

A user interface to the **portmap** service, which destroys all mapping between the triple [*prognum*,*versnum*,*] and **ports** on the machine's **portmap** service. This routine returns one if it succeeds, zero otherwise.

**registerrpc(prognum, versnum, procnum, procname, inproc, outproc)**
**u_long prognum, versnum, procnum;**
**char *(*procname) () ;**
**xdrproc_t inproc, outproc;**

Register procedure *procname* with the RPC service package. If a request arrives for program *prognum*, version *versnum*, and procedure *procnum*, *procname* is called with a pointer to its parameter(s); *progname* should return a pointer to its static result(s); *inproc* is used to decode the parameters while *outproc* is used to encode the results. This routine returns zero if the registration succeeded, −1 otherwise.

Warning: remote procedures registered in this form are accessed using the UDP/IP transport; see **svcudp_create()** for restrictions.

**struct rpc_createerr rpc_createerr;**

A global variable whose value is set by any RPC client creation routine that does not succeed. Use the routine **clnt_pcreateerror()** to print the reason why.

**svc_destroy(xprt)**
**SVCXPRT ***
**xprt;**

A macro that destroys the RPC service transport handle, *xprt*. Destruction usually involves deallocation of private data structures, including *xprt* itself. Use of *xprt* is undefined after calling this routine.

**fd_set svc_fdset;**

A global variable reflecting the RPC service side's read file descriptor bit mask; it is suitable as a parameter to the **select** system call. This is only of interest if a service implementor does not call **svc_run()**, but rather does his own asynchronous event processing. This variable is read-only (do not pass its address to **select**!), yet it may change after calls to **svc_getreqset()** or any creation routines.

**int svc_fds;**

Similar to **svc_fdset()**, but limited to 32 descriptors. This interface is obsoleted by **svc_fdset()**.

**svc_freeargs(xprt, inproc, in)**
**SVCXPRT *xprt;**
**xdrproc_t inproc;**
**char *in;**

A macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using **svc_getargs()**. This routine returns 1 if the results were successfully freed, and zero otherwise.

**svc_getargs(xprt, inproc, in)**
**SVCXPRT *xprt;**
**xdrproc_t inproc;**
**char *in;**

A macro that decodes the arguments of an RPC request associated with the RPC service transport handle, *xprt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns one if decoding succeeds, and zero otherwise.

**struct sockaddr_in ***
**svc_getcaller(xprt)**
**SVCXPRT *xprt;**

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle, *xprt*.

**svc_getreqset(rdfds)**
**fd_set *rdfds;**

This routine is only of interest if a service implementor does not call **svc_run()**, but instead implements custom asynchronous event processing. It is called when the **select** system call has determined that an RPC request has arrived on some RPC **socket(s)** ; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all sockets associated with the value of *rdfds* have been serviced.

**svc_getreq(rdfds)**
**int rdfds;**

Similar to **svc_getreqset()**, but limited to 32 descriptors. This interface is obsoleted by **svc_getreqset()**.

**svc_register(xprt, prognum, versnum, dispatch, protocol)**
**SVCXPRT *xprt;**
**u_long prognum, versnum;**
**void (*dispatch) ();**
**u_long protocol;**

Associates *prognum* and *versnum* with the service dispatch procedure, *dispatch*. If *protocol* is zero, the service is not registered with the **portmap** service. If *protocol* is non-zero, then a mapping of the triple [*prognum*,*versnum*,*protocol*] to **xprt->xp_port** is established with the local **portmap** service (generally *protocol* is zero, **IPPROTO_UDP** or **IPPROTO_TCP** ). The procedure *dispatch* has the following form:

**dispatch(request, xprt)**
**struct svc_req *request;**
**SVCXPRT *xprt;**

The **svc_register()** routine returns one if it succeeds, and zero otherwise.

**svc_run()**

This routine never returns. It waits for RPC requests to arrive, and calls the appropriate service procedure using **svc_getreq()** when one arrives. This procedure is usually waiting for a **select()** system call to return.

**svc_sendreply(xprt, outproc, out)**
**SVCXPRT *xprt;**
**xdrproc_t outproc;**
**char *out;**

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns one if it succeeds, zero otherwise.

**void**
**svc_unregister(prognum, versnum)**
**u_long prognum, versnum;**

Remove all mapping of the double [*prognum*,*versnum*] to dispatch routines, and of the triple [*prognum*,*versnum*,*] to port number.

**void**
**svcerr_auth(xprt, why)**
**SVCXPRT *xprt;**
**enum auth_stat why;**

Called by a service dispatch routine that refuses to perform a remote procedure call due to an authentication error.

**void**
**svcerr_decode(xprt)**
**SVCXPRT *xprt;**

Called by a service dispatch routine that cannot successfully decode its parameters. See also **svc_getargs()**.

**void**
**svcerr_noproc(xprt)**
**SVCXPRT *xprt;**

Called by a service dispatch routine that does not implement the procedure number that the caller requests.

**void**
**svcerr_noprog(xprt)**
**SVCXPRT *xprt;**

Called when the desired program is not registered with the RPC package. Service implementors usually do not need this routine.

**void**
**svcerr_progvers(xprt)**
**SVCXPRT *xprt;**

Called when the desired version of a program is not registered with the RPC package. Service implementors usually do not need this routine.

**void**
**svcerr_systemerr(xprt)**
**SVCXPRT *xprt;**

Called by a service dispatch routine when it detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.

**void**
**svcerr_weakauth(xprt)**
**SVCXPRT *xprt;**

Called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient (but correct) authentication parameters. The routine calls **svcerr_auth(xprt, AUTH_TOOWEAK)**.

**SVCXPRT ***
**svcraw_create()**

This routine creates a toy RPC service transport, to which it returns a pointer. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; see **clntraw_create()**. This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel interference. This routine returns NULL if it fails.

**SVCXPRT ***
**svctcp_create(sock, send_buf_size, recv_buf_size)**
**int sock;**
**u_int send_buf_size, recv_buf_size;**

This routine creates a TCP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the socket *sock*, which may be **RPC_ANYSOCK**, in which case a new socket is created. If the socket is not bound to a local TCP port, then this routine binds it to an arbitrary port. Upon completion, **xprt->xp_sock** is the transport's socket descriptor, and **xprt->xp_port** is the transport's port number. This routine returns NULL if it fails. Since TCP-based RPC uses buffered I/O , users may specify the size of buffers; values of zero choose suitable defaults.

**void**
**svcfd_create(fd, sendsize, recvsize)**
**int fd;**
**u_int sendsize;**
**u_int recvsize;**

Create a service on top of any open descriptor. Typically, this descriptor is a connected socket for a stream protocol such as TCP. *sendsize* and *recvsize* indicate sizes for the send and receive buffers. If they are zero, a reasonable default is chosen.

**SVCXPRT ***
**svcudp_create(sock)**
**int sock;**

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the socket *sock*, which may be **RPC_ANYSOCK** , in which case a new socket is created. If the socket is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, **xprt->xp_sock** is the transport's socket descriptor, and **xprt->xp_port** is the transport's port number. This routine returns NULL if it fails.

Warning: since UDP-based RPC messages can only hold up to 8 Kbytes of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

**user2netname(name, uid, domain)**
**char *name;**
**int uid;**
**char *domain;**

Convert from a domain-specific username to an operating-system independent netname. Returns TRUE if it succeeds and FALSE if it fails. Inverse of **netname2user()**.

**xdr_accepted_reply(xdrs, ar)**
**XDR *xdrs;**
**struct accepted_reply *ar;**

Used for encoding RPC reply messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

**xdr_authunix_parms(xdrs, aupp)**
**XDR *xdrs;**
**struct authunix_parms *aupp;**

Used for describing UNIX credentials. This routine is useful for users who wish to generate these credentials without using the RPC authentication package.

**void**
**xdr_callhdr(xdrs, chdr)**
**XDR *xdrs;**
**struct rpc_msg *chdr;**

Used for describing RPC call header messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

**xdr_callmsg(xdrs, cmsg)**
**XDR *xdrs;**
**struct rpc_msg *cmsg;**

Used for describing RPC call messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

**xdr_opaque_auth(xdrs, ap)**
**XDR *xdrs;**
**struct opaque_auth *ap;**

Used for describing RPC authentication information messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

**xdr_pmap(xdrs, regs)**
**XDR *xdrs;**
**struct pmap *regs;**

Used for describing parameters to various **portmap** procedures, externally. This routine is useful for users who wish to generate these parameters without using the **pmap** interface.

**xdr_pmaplist(xdrs, rp)**
**XDR *xdrs;**
**struct pmaplist **rp;**

Used for describing a list of port mappings, externally. This routine is useful for users who wish to generate these parameters without using the **pmap** interface.

**xdr_rejected_reply(xdrs, rr)**
**XDR *xdrs;**
**struct rejected_reply *rr;**

Used for describing RPC reply messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

**xdr_replymsg(xdrs, rmsg)**
**XDR *xdrs;**
**struct rpc_msg *rmsg;**

Used for describing RPC reply messages. This routine is useful for users who wish to generate RPC style messages without using the RPC package.

**void**
**xprt_register(xprt)**
**SVCXPRT *xprt;**

After RPC service transport handles are created, they should register themselves with the RPC service package. This routine modifies the global variable **svc_fds()**. Service implementors usually do not need this routine.

**void**
**xprt_unregister(xprt)**
**SVCXPRT *xprt;**

Before an RPC service transport handle is destroyed, it should unregister itself with the RPC service package. This routine modifies the global variable **svc_fds()**. Service implementors usually do not need this routine.

SEE ALSO

**xdr**(3N), **keyserv**(8C)

*Network Programming*:

NAME

rtime – get remote time

SYNOPSIS

```
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>

int rtime(addrp, timep, timeout)
struct sockaddr_in *addrp;
struct timeval *timep;
struct timeval *timeout;
```

DESCRIPTION

**rtime**( ) consults the Internet Time Server at the address pointed to by *addrp* and returns the remote time in the **timeval** struct pointed to by *timep*. Normally, the UDP protocol is used when consulting the Time Server. The *timeout* parameter specifies how long the routine should wait before giving up when waiting for a reply. If *timeout* is specified as NULL, however, the routine will instead use TCP and block until a reply is received from the time server.

The routine returns 0 if it is successful. Otherwise, it returns −1 and **errno** is set to reflect the cause of the error.

SEE ALSO

**timed**(8C)

NAME

scandir, alphasort – scan a directory

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>

scandir(dirname, namelist, select, compar)
char *dirname;
struct direct **namelist;
int (*select)();
int (*compar)();

alphasort(d1, d2)
struct direct **d1, **d2;
```

DESCRIPTION

**scandir()** reads the directory **dirname** and builds an array of pointers to directory entries using **malloc**(3). The second parameter is a pointer to an array of structure pointers. The third parameter is a pointer to a routine which is called with a pointer to a directory entry and should return a non zero value if the directory entry should be included in the array. If this pointer is NULL, then all the directory entries will be included. The last argument is a pointer to a routine which is passed to **qsort**(3) to sort the completed array. If this pointer is NULL, the array is not sorted. **alphasort()** is a routine which will sort the array alphabetically.

**scandir()** returns the number of entries in the array and a pointer to the array through the parameter *namelist*.

SEE ALSO

**directory**(3), **malloc**(3), **qsort**(3)

DIAGNOSTICS

Returns –1 if the directory cannot be opened for reading or if **malloc**(3) cannot allocate enough memory to hold all the data structures.

## NAME

scanf, fscanf, sscanf – formatted input conversion

## SYNOPSIS

```
#include <stdio.h>

scanf(format [ , pointer ] ... )
char *format;

fscanf(stream, format [ , pointer ] ... )
FILE *stream;
char *format;

sscanf(s, format [ , pointer ] ... )
char *s, *format;
```

## DESCRIPTION

**scanf()** reads from the standard input stream **stdin**. **fscanf()** reads from the named input stream. **sscanf()** reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format*, described below, and a set of *pointer* arguments indicating where the converted input should be stored. The results are undefined in there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (SPACE, TAB, or NEWLINE) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not '**%**'), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character '**%**', an optional assignment suppressing character '*****', an optional numerical maximum field width, an optional **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by '*****'. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except "[" and "c", white space leading an input field is ignored.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion characters are legal:

- **%** A single **%** is expected in the input at this point; no assignment is done.
- **d** A decimal integer is expected; the corresponding argument should be an integer pointer.
- **u** An unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.
- **o** An octal integer is expected; the corresponding argument should be an integer pointer.
- **x** A hexadecimal integer is expected; the corresponding argument should be an integer pointer.
- **i** An integer is expected; the corresponding argument should be an integer pointer. It will store the value of the next input item interpreted according to C conventions: a leading "0" implies octal; a leading "0x" implies hexadecimal; otherwise, decimal.
- **n** Stores in an integer argument the total number of characters (including white space) that have been scanned so far since the function call. No input is consumed.

**e,f,g** A floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is as described for **string_to_decimal**(3), with *fortran_exponent* zero.

**s** A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a white space character.

**c** A character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use **%1s**. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

**[** Indicates string data; the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first–last*, thus [0123456789] may be expressed [0–9]. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating **\0**, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters **d**, **u**, **o**, **x**, and **i** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list. The **l** or **h** modifier is ignored for other conversion characters.

*Avoid this common error:* because **printf**(3S) does not require that the lengths of conversion descriptors and actual parameters match, coders sometimes are careless with the **scanf**( ) functions. But converting %f to &double or %lf to &float *does not work*; the results are quite incorrect.

**scanf**( ) conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

**scanf**( ) returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. The constant EOF is returned upon end of input. Note: this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

If the input ends before the first conflict or conversion, EOF is returned. If the input ends after the first conflict or conversion, the number of successfully matched items is returned.

## EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to *n* the value **3**, to *i* the value **25**, to *x* the value **5.432**, and *name* will contain **thompson\0**. Or:

```
int i, j; float x; char name[50];
(void) scanf("%i%2d%f%*d %[0–9]", &j, &i, &x, name);
```

with input:

```
011 56789 0123 56a72
```

will assign **9** to *j*, **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name*. The next call to **getchar( )** (see **getc**(3S)) will return **a**. Or:

```
int i, j, s, e; char name[50];
(void) scanf("%i %i %n%s%n", &i, &j, &s, name, &e);
```

with input:

```
0x11 0xy johnson
```

will assign **17** to *i*, **0** to *j*, **6** to *s*, will place the string **xy\0** in *name*, and will assign **8** to *e*. Thus, the length of *name* is $e - s = 2$. The next call to **getchar( )** (see **getc**(3S)) will return a SPACE.

## SEE ALSO

**getc**(3S), **printf**(3S), **scanf**(3V), **stdio**(3S), **string_to_decimal**(3), **strtol**(3)

## DIAGNOSTICS

These functions return EOF on end of input, and a short count for missing or illegal data items.

## BUGS

The success of literal matches and suppressed assignments is not directly determinable.

## WARNINGS

Trailing white space (including a NEWLINE) is left unread unless matched in the control string.

NAME

setbuf, setbuffer, setlinebuf, setvbuf – assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;

setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

setlinebuf(stream)
FILE *stream;

int setvbuf(stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;
```

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a NEWLINE is encountered or input is read from **stdin**. **fflush**() (see **fclose**(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from **malloc**(3) upon the first **getc**() or **putc**(3S) on the file. If the standard stream **stdout** refers to a terminal it is line buffered. The standard stream **stderr** is unbuffered by default.

**setbuf**() can be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered. A manifest constant **BUFSIZ**, defined in the **<stdio.h>** header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

**setbuffer**, an alternate form of **setbuf**, can be used after a stream has been opened but before it is read or written. It uses the character array *buf* whose size is determined by the *size* argument instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered.

**setvbuf**() can be used after a stream has been opened but before it is read or written. *type* determines how stream will be buffered. Legal values for *type* (defined in **<stdio.h>**) are:

| | |
|---|---|
| **_IOFBF** | fully buffers the input/output. |
| **_IOLBF** | line buffers the output; the buffer will be flushed when a NEWLINE is written, the buffer is full, or input is requested. |
| **_IONBF** | completely unbuffers the input/output. |

If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *size* specifies the size of the buffer to be used.

**setlinebuf**() is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike **setbuf**, **setbuffer**, and **setvbuf**, it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using **freopen()** (see **fopen**(3S)). A file can be changed from block buffered or line buffered to unbuffered by using **freopen()** followed by **setbuf()** with a buffer argument of NULL.

NOTE

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

SEE ALSO

**fclose**(3S), **fopen**(3S), **fread**(3S), **getc**(3S), **malloc**(3), **printf**(3S), **putc**(3S), **puts**(3S), **setbuf**(3V)

DIAGNOSTICS

If an illegal value for *type* or *size* is provided, **setvbuf()** returns a non-zero value. Otherwise, the value returned will be zero.

NAME

setjmp, longjmp, sigsetjmp, siglongjmp – non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;
int val;

int _setjmp(env)
jmp_buf env;

_longjmp(env, val)
jmp_buf env;
int val;

int sigsetjmp(env, savemask)
sigjmp_buf env;
int savemask;

siglongjmp(env, val)
sigjmp_buf env;
int val;
```

DESCRIPTION

**setjmp**( ) and **longjmp**( ) are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

**setjmp**( ) saves its stack environment in *env* for later use by **longjmp**. A normal call to **setjmp**( ) returns zero. **setjmp**( ) also saves the register environment. If a **longjmp**( ) call will be made, the routine which called **setjmp**( ) should not return until after the **longjmp**( ) has returned control (see below).

**longjmp**( ) restores the environment saved by the last call of **setjmp**, and then returns in such a way that execution continues as if the call of **setjmp**( ) had just returned the value *val* to the function that invoked **setjmp**; however, if *val* were zero, execution would continue as if the call of **setjmp**( ) had returned one. This ensures that a "return" from **setjmp**( ) caused by a call to **longjmp**( ) can be distinguished from a regular return from **setjmp**. The calling function must not itself have returned in the interim, otherwise **longjmp**( ) will be returning control to a possibly non-existent environment. All memory-bound data have values as of the time **longjmp**( ) was called. The CPU and floating-point data registers are restored to the values they had at the time that **setjmp**( ) was called. But, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp**. This is especially a problem for programmers trying to write machine-independent C routines.

**setjmp**( ) and **longjmp**( ) save and restore the signal mask (see **sigsetmask**(2)), while **_setjmp** and **_longjmp** manipulate only the C stack and registers. If the *savemask* flag to **sigsetjmp** is non-zero, the signal mask is saved, and a subsequent **siglongjmp** using the same *env* will restore the signal mask. If the *savemask* flag is zero, the signal mask is not saved, and a subsequent **siglongjmp** using the same *env* will not restore the signal mask. In all other ways, **_setjmp** and **sigsetjmp** function in the same way that **setjmp**( ) does, and **_longjmp** and **siglongjmp** function in the same way that **longjmp**( ) does.

None of these functions save or restore any floating-point status or control registers, in particular the MC68881 **fpsr**, **fpcr**, or **fpiar**, the Sun-3 FPA **fpamode** or **fpastatus**, and the Sun-4 **%fsr**. See **ieee_flags**(3M) to save and restore floating-point status or control information.

EXAMPLE

The following code fragment indicates the flow of control of the **setjmp()** and **longjmp()** combination:

```
function declaration
...
        jmp_buf  my_environment;
          ...
        if (setjmp (my_environment)) {
          /* register variables have unpredictable values
          code after the return from longjmp
                  ...
        } else {
          /* do not modify register vars
          this is the return from setjmp
                  ...
        }
```

SEE ALSO

**cc**(1V), **sigsetmask**(2), **sigvec**(2), **ieee_flags**(3M), **signal**(3), **setjmp**(3V)

BUGS

**setjmp()** does not save the current notion of whether the process is executing on the signal stack. The result is that a **longjmp()** to some place on the signal stack leaves the signal stack state incorrect.

On Sun-2 and Sun-3 systems **setjmp()** also saves the register environment. Therefore, all data that are bound to registers are restored to the values they had at the time that **setjmp()** was called. All memory-bound data have values as of the time **longjmp()** was called. However, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp**. When using compiler options that specify automatic register allocation (see **cc**(1V)), the compiler will not attempt to assign variables to registers in routines that call **setjmp**.

**longjmp()** never causes **setjmp()** to return zero in the Sun implementation; this is also true of many other implementations, including all System V implementations, so programmers should not depend on **longjmp()** being able to cause **setjmp()** to return zero.

## NAME

setuid, seteuid, setruid, setgid, setegid, setrgid – set user and group ID

## SYNOPSIS

**setuid(uid)**
**seteuid(euid)**
**setruid(ruid)**

**setgid(gid)**
**setegid(egid)**
**setrgid(rgid)**

## DESCRIPTION

**setuid()** (**setgid**) sets both the real and effective user ID (group ID) of the current process to as specified.

**seteuid()** (**setegid**) sets the effective user ID (group ID) of the current process.

**setruid()** (**setrgid**) sets the real user ID (group ID) of the current process.

These calls are only permitted to the super-user or if the argument is the real or effective ID.

## SEE ALSO

**getgid**(2), **getuid**(2), **setregid**(2), **setreuid**(2)

## DIAGNOSTICS

Zero is returned if the user (group) ID is set; −1 is returned otherwise, with the global variable **errno** set as for **setreuid()** or **setregid**.

NAME

sigfpe - signal handling for specific SIGFPE codes

SYNOPSIS

**#include <signal.h>**

**#include <floatingpoint.h>**

**sigfpe_handler_type sigfpe(code, hdl)**
**sigfpe_code_type code;**
**sigfpe_handler_type hdl;**

DESCRIPTION

This function allows signal handling to be specified for particular SIGFPE codes. A call to **sigfpe()** defines a new handler *hdl* for a particular SIGFPE *code* and returns the old handler as the value of the function **sigfpe.** Normally handlers are specified as pointers to functions; the special cases **SIGFPE_IGNORE, SIGFPE_ABORT,** and **SIGFPE_DEFAULT** allow ignoring, specifying core dump using **abort**(3), or default handling respectively.

For these IEEE-related codes:

| | |
|---|---|
| **FPE_FLTINEX_TRAP** | fp_inexact - floating inexact result |
| **FPE_FLTDIV_TRAP** | fp_division - floating division by zero |
| **FPE_FLTUND_TRAP** | fp_underflow - floating underflow |
| **FPE_FLTOVF_TRAP** | fp_overflow - floating overflow |
| **FPE_FLTBSUN_TRAP** | fp_invalid - branch or set on unordered |
| **FPE_FLTOPERR_TRAP** | fp_invalid - floating operand error |
| **FPE_FLTNAN_TRAP** | fp_invalid - floating Not-A-Number |

default handling is defined to be to call the handler specified to **ieee_handler**(3M).

For all other SIGFPE codes, default handling is to core dump using **abort**(3).

The compilation option –ffpa causes fpa recomputation to replace the default abort action for code **FPE_FPA_ERROR.** Note: **SIGFPE_DEFAULT** will restore abort rather than FPA recomputation for this code.

Three steps are required to intercept an IEEE-related SIGFPE code with **sigfpe**:

1) Set up a handler with **sigfpe.**

2) Enable the relevant IEEE trapping capability in the hardware, perhaps by using assembly-language instructions.

3) Perform a floating-point operation that generates the intended IEEE exception.

Unlike **ieee_handler**(3M), **sigfpe()** never changes floating-point hardware mode bits affecting IEEE trapping. No IEEE-related SIGFPE signals will be generated unless those hardware mode bits are enabled.

SIGFPE signals can be handled using **sigvec**(2), **signal**(3), **sigfpe**(3), or **ieee_handler**(3M). In a particular program, to avoid confusion, use only one of these interfaces to handle SIGFPE signals.

**EXAMPLE**

A user-specified signal handler might look like this:

```
void sample_handler( sig, code, scp, addr )
        int sig ;           /* sig == SIGFPE always */
        int code ;
        struct sigcontext *scp ;
        char *addr ;
        {
                /*
                  Sample user-written sigfpe code handler.
                  Prints a message and continues.
                  struct sigcontext is defined in <signal.h>.
                 */
                printf(" ieee exception code %x occurred at pc %X \n",code,scp->sc_pc);
        }
```

and it might be set up like this:

```
        extern void sample_handler();
        main()
        {
                sigfpe_handler_type hdl, old_handler1, old_handler2;
        /*
         * save current overflow and invalid handlers; set the new
         * overflow handler to sample_handler() and set the new
         * invalid handler to SIGFPE_ABORT (abort on invalid)
         */
                hdl = (sigfpe_handler_type) sample_handler;
                old_handler1 = sigfpe(FPE_FLTOVF_TRAP, hdl);
                old_handler2 = sigfpe(FPE_FLTOPERR_TRAP, SIGFPE_ABORT);
                ...
        /*
         * restore old overflow and invalid handlers
         */
                sigfpe(FPE_FLTOVF_TRAP,   old_handler1);
                sigfpe(FPE_FLTOPERR_TRAP, old_handler2);
        }
```

**FILES**

**/usr/include/floatingpoint.h**
**/usr/include/signal.h**

**SEE ALSO**

**sigvec**(2), **abort**(3), **floatingpoint**(3), **ieee_handler**(3M), **signal**(3),

**DIAGNOSTICS**

**sigfpe( )** returns BADSIG if *code* is not zero or a defined SIGFPE code.

NAME

siginterrupt – allow signals to interrupt system calls

SYNOPSIS

**int siginterrupt(sig, flag)**
**int sig, flag;**

DESCRIPTION

**siginterrupt()** is used to change the system call restart behavior when a system call is interrupted by the specified signal. If the flag is false (0), then system calls will be restarted if they are interrupted by the specified signal and no data has been transferred yet. System call restart is the default behavior on 4.2BSD, and on SunOS in the 4.2 environment, when the **signal (3)** routine is used.

If the flag is true (1), then restarting of system calls is disabled. If a system call is interrupted by the specified signal and no data has been transferred, the system call will return –1 with **errno** set to EINTR. Interrupted system calls that have started transferring data will return the amount of data actually transferred. System call interrupt is the signal behavior found on older version of the UNIX operating systems, such as 4.1BSD and System V UNIX. It is the default behavior on SunOS in the System V environment when the **signal()** routine is used; therefore, this routine is useful in that environment only if a signal that a sigvec(2) specified should restart system calls is to be changed not to restart them.

Note: the new 4.2BSD signal handling semantics are not altered in any other way. Most notably, signal handlers always remain installed until explicitly changed by a subsequent **sigvec** call, and the signal mask operates as documented in **sigvec**, unless the **SV_RESETHAND** bit has been used to specify that the pre-4.2BSD signal behavior is to be used. Programs may switch between restartable and interruptible system call operation as often as desired in the execution of a program.

Issuing a **siginterrupt()** call during the execution of a signal handler will cause the new action to take place on the next signal to be caught.

NOTES

This library routine uses an extension of the **sigvec**(2) system call that is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

RETURN VALUE

A 0 value indicates that the call succeeded. A –1 value indicates that an invalid signal number has been supplied.

SEE ALSO

**sigblock**(2), **sigpause**(2), **sigsetmask**(2), **sigvec**(2), **signal**(3)

## NAME

signal – simplified software signal facilities

## SYNOPSIS

**#include <signal.h>**

**void (*signal(sig, func))()**
**void (*func)();**

## DESCRIPTION

**signal()** is a simplified interface to the more general **sigvec**(2) facility. Programs that use **signal()** in preference to **sigvec()** are more likely to be portable to all systems.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see **termio**(4)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the **signal()** call allows signals either to be ignored or to interrupt to a specified location. The following is a list of all signals with names as in the include file **<signal.h>**:

| | | |
|---|---|---|
| **SIGHUP** | 1 | hangup |
| **SIGINT** | 2 | interrupt |
| **SIGQUIT** | 3* | quit |
| **SIGILL** | 4* | illegal instruction |
| **SIGTRAP** | 5* | trace trap |
| **SIGABRT** | 6* | abort (generated by **abort**(3) routine) |
| **SIGEMT** | 7* | emulator trap |
| **SIGFPE** | 8* | arithmetic exception |
| **SIGKILL** | 9 | kill (cannot be caught, blocked, or ignored) |
| **SIGBUS** | 10* | bus error |
| **SIGSEGV** | 11* | segmentation violation |
| **SIGSYS** | 12* | bad argument to system call |
| **SIGPIPE** | 13 | write on a pipe or other socket with no one to read it |
| **SIGALRM** | 14 | alarm clock |
| **SIGTERM** | 15 | software termination signal |
| **SIGURG** | 16• | urgent condition present on socket |
| **SIGSTOP** | 17† | stop (cannot be caught, blocked, or ignored) |
| **SIGTSTP** | 18† | stop signal generated from keyboard |
| **SIGCONT** | 19• | continue after stop (cannot be blocked) |
| **SIGCHLD** | 20• | child status has changed |
| **SIGTTIN** | 21† | background read attempted from control terminal |
| **SIGTTOU** | 22† | background write attempted to control terminal |
| **SIGIO** | 23• | I/O is possible on a descriptor (see **fcntl**(2V)) |
| **SIGXCPU** | 24 | cpu time limit exceeded (see **getrlimit**(2)) |
| **SIGXFSZ** | 25 | file size limit exceeded (see **getrlimit**(2)) |
| **SIGVTALRM** | 26 | virtual time alarm (see **getitimer**(2)) |
| **SIGPROF** | 27 | profiling timer alarm (see **getitimer**(2)) |
| **SIGWINCH** | 28• | window changed (see **termio**(4) and **win**(4S)) |
| **SIGLOST** | 29* | resource lost (see **lockd**(8C)) |
| **SIGUSR1** | 30 | user-defined signal 1 |
| **SIGUSR2** | 31 | user-defined signal 2 |

The starred signals in the list above cause a core image if not caught or ignored.

If *func* is **SIG_DFL**, the default action for signal *sig* is reinstated; this default is termination (with a core image for starred signals) except for signals marked with • or †. Signals marked with • are discarded if the action is **SIG_DFL**; signals marked with † cause the process to stop. If *func* is **SIG_IGN** the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurrences of the signal are automatically blocked and *func* is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted. **Unlike previous signal facilities, the handler *func* remains installed after a signal has been delivered.**

If a caught signal occurs during certain system calls, terminating the call prematurely, the call is automatically restarted. In particular this can occur during a **read**(2V) or **write**(2V) on a slow device (such as a terminal; but not a file) and during a **wait**(2).

The value of **signal()** is the previous (or initial) value of *func* for the particular signal.

After a **fork**(2) or **vfork**(2) the child inherits all signals. An **execve**(2) resets all caught signals to the default action; ignored signals remain ignored.

## NOTES

The handler routine can be declared:

```
void handler(sig, code, scp, addr)
int sig, code;
struct sigcontext *scp;
char *addr;
```

Here *sig* is the signal number; *code* is a parameter of certain signals that provides additional detail; *scp* is a pointer to the **sigcontext** structure (defined in **<signal.h>**), used to restore the context from before the signal; and *addr* is additional address information. See **sigvec**(2) for more details.

## RETURN VALUE

The previous action is returned on a successful call. Otherwise, −1 is returned and **errno** is set to indicate the error.

## ERRORS

**signal()** will fail and no action will take place if one of the following occur:

| | |
|---|---|
| EINVAL | *sig* is not a valid signal number. |
| EINVAL | An attempt is made to ignore or supply a handler for **SIGKILL** or **SIGSTOP**. |
| EINVAL | An attempt is made to ignore **SIGCONT** (by default **SIGCONT** is ignored). |

## SEE ALSO

**kill**(1), **execve**(2), **fork**(2), **getitimer**(2), **getrlimit**(2), **kill**(2V), **ptrace**(2), **read**(2V), **sigblock**(2), **sigpause**(2), **sigsetmask**(2), **sigstack**(2), **sigvec**(2), **vfork**(2), **wait**(2), **write**(2V), **setjmp**(3), **termio**(4)

NAME

sleep – suspend execution for interval

SYNOPSIS

**sleep(seconds)**
**unsigned seconds;**

DESCRIPTION

**sleep( )** suspends the current process from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and may be an arbitrary amount longer because of other activity in the system.

**sleep( )** is implemented by setting an interval timer and pausing until it expires. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous value of the timer, the process sleeps only until the timer would have expired, and the signal which occurs with the expiration of the timer is sent one second later.

SEE ALSO

**getitimer**(2), **sigpause**(2), **usleep**(3)

NAME

ssignal, gsignal – software signals

SYNOPSIS

**#include <signal.h>**

**int (*ssignal (sig, action))()**
**int sig, (*action)();**

**int gsignal (sig)**
**int sig;**

DESCRIPTION

**ssignal()** and **ssignal()** implement a software facility similar to **signal**(3).

Software signals made available to users are associated with integers in the inclusive range 1 through 15. A call to **ssignal()** associates a procedure, *action*, with the software signal *sig*; the software signal, *sig*, is raised by a call to **ssignal**. Raising a software signal causes the action established for that signal to be *taken*.

The first argument to **ssignal()** is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user-defined) *action function* or one of the manifest constants **SIG_DFL (default)**or **SIG_IGN (ignore). ssignal()** returns the action previously established for that signal type; if no action has been established or the signal number is illegal, **ssignal()** returns **SIG_DFL.**

**ssignal()** raises the signal identified by its argument, *sig*:

If an action function has been established for *sig*, then that action is reset to **SIG_DFL** and the action function is entered with argument *sig*. **ssignal()** returns the value returned to it by the action function.

If the action for *sig* is **SIG_IGN**, **ssignal()** returns the value 1 and takes no other action.

If the action for *sig* is **SIG_DFL**, **ssignal()** returns the value 0 and takes no other action.

If *sig* has an illegal value or no action was ever specified for *sig*, **ssignal()** returns the value 0 and takes no other action.

SEE ALSO

**signal**(3)

## NAME

stdio – standard buffered input/output package

## SYNOPSIS

**#include <stdio.h>**

**FILE *stdin;**
**FILE *stdout;**
**FILE *stderr;**

## DESCRIPTION

The functions described in section 3S constitute a user-level I/O buffering scheme. The in-line macros **getc**(3S) and **putc**(3S) handle characters quickly. The macros **getchar** and **putchar**, and the higher level routines **fgetc, getw, gets, fgets, scanf, fscanf, fread, fputc, putw, puts, fputs, printf, fprintf, fwrite** all use or act as if they use **getc**( ) and **putc**( ) ; they can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type **FILE**. **fopen**(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the **<stdio.h>** include file and associated with the standard open files:

| | |
|---|---|
| **stdin** | standard input file |
| **stdout** | standard output file |
| **stderr** | standard error file |

A constant **NULL** (0) designates a nonexistent pointer.

An integer constant **EOF** (–1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

Any module that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in sections labeled 3S of this manual are declared in that header file and need no further declaration. The constants and the following 'functions' are implemented as macros; redeclaration of these names is perilous: **getc, getchar, putc, putchar, feof, ferror, fileno**, and **clearerr**.

Output streams, with the exception of the standard error stream **stderr**, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream **stderr** is by default unbuffered, but use of **fopen**(3S) will cause it to become buffered or line-buffered. When an output stream is unbuffered, information is written to the destination file or terminal as soon as it is output to the stream; when it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is written to the destination file or terminal as soon as the line is completed (that is, as soon as a NEWLINE character is output or, if the output stream is **stdout** or **stderr**, as soon as input is read from **stdin**). **setbuf**(3S), **setbuffer**, **setlinebuf**, or **setvbuf** can be used to change the stream's buffering strategy.

## SEE ALSO

**open**(2V), **close**(2), **lseek**(2), **pipe**(2), **read**(2V), **write**(2V), **ctermid**(3S), **cuserid**(3S), **fclose**(3S), **ferror**(3S), **fopen**(3S), **fread**(3S), **fseek**(3S), **getc**(3S), **gets**(3S), **popen**(3S), **printf**(3S), **putc**(3S), **puts**(3S), **scanf**(3S), **setbuf**(3S), **system**(3), **tmpfile**(3S), **tmpnam**(3S), **ungetc**(3S)

## DIAGNOSTICS

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with **fopen**, input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

BUGS

The standard buffered functions do not interact well with certain other library and system functions, especially **vfork**(2).

NOTES

The line buffering of output to terminals is almost always transparent, but may cause confusion or malfunctioning of programs which use standard I/O routines but use **read**(2V) to read from the standard input, as calls to **read**() do not cause output to line-buffered streams to be flushed.

In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to call **fflush** (see **fclose**(3S)) on the standard output before performing the computation so that the output will appear.

NAME

string, strcat, strncat, strdup, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok, index, rindex – string operations

SYNOPSIS

```
#include <string.h>

char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;
int n;

char *strdup(s1)
char *s1;

int strcmp(s1, s2)
char *s1, *s2;

int strncmp(s1, s2, n)
char *s1, *s2;
int n;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;
int n;

int strlen(s)
char *s;

char *strchr(s, c)
char *s;
int c;

char *strrchr(s, c)
char *s;
int c;

char *strpbrk(s1, s2)
char *s1, *s2;

int strspn(s1, s2)
char *s1, *s2;

int strcspn(s1, s2)
char *s1, *s2;

char *strtok(s1, s2)
char *s1, *s2;

#include <strings.h>

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s, c;
```

DESCRIPTION

These functions operate on NULL-terminated strings. They do not check for overflow of any receiving string.

**strcat()** appends a copy of string *s2* to the end of string *s1*. **strncat()** appends at most *n* characters. Each returns a pointer to the NULL-terminated result.

**strcmp()** compares its arguments and returns an integer greater than, equal to, or less than 0, according as *s1* is lexicographically greater than, equal to, or less than *s2*. **strncmp()** makes the same comparison but compares at most *n* characters.

**strdup()** returns a pointer to a new string which is a duplicate of the string pointed to by *s1*. The space for the new string is obtained using **malloc**(3). If the new string cannot be created, a NULL pointer is returned.

**strcpy()** copies string *s2* to *s1*, stopping after the NULL character has been copied. **strncpy()** copies exactly *n* characters, truncating or NULL-padding *s2*. The result will not be NULL-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

**strlen()** returns the number of characters in *s*, not including the NULL-terminating character.

**strchr()** (**strrchr**) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The NULL character terminating a string is considered to be part of the string.

**index()** (**rindex**) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. These functions are identical to **strchr()** (**strchr**) and merely have different names.

**strpbrk()** returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.

**strspn()** (**strcspn**) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

**strtok()** considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a NULL character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument a NULL pointer) will work through the string *s1* immediately following that token. In this way subsequent calls will work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a NULL pointer is returned.

NOTE

For user convenience, all these functions, except for **index()** and **rindex**, are declared in the optional **<string.h>** header file. All these functions, including **index()** and **rindex()** but excluding **strchr**, **strrchr**, **strpbrk**, **strspn**, **strcspn**, and **strtok**, are declared in the optional **<strings.h>** include file; these headers are set this way for backward compatibility.

SEE ALSO

**malloc**(3), **bstring**(3)

WARNINGS

**strcmp()** and **strncmp()** use native character comparison, which is signed on the Sun, but may be unsigned on other machines. Thus the sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

On the Sun processor, as well as on many other machines, you can *not* use a NULL pointer to indicate a NULL string. A NULL pointer is an error and results in an abort of the program. If you wish to indicate a NULL string, you must have a pointer that points to an explicit NULL string. On some implementations of the C language on some machines, a NULL pointer, if dereferenced, would yield a NULL string; this highly non-portable trick was used in some programs. Programmers using a NULL pointer to represent an empty string should be aware of this portability issue; even on machines where dereferencing a NULL pointer does not cause an abort of the program, it does not necessarily yield a

NULL string.

Character movement is performed differently in different implementations. Thus overlapping moves may yield surprises.

NAME

string_to_decimal, file_to_decimal, func_to_decimal – parse characters into decimal record

SYNOPSIS

```
#include <floatingpoint.h>
#include <stdio.h>

void string_to_decimal(pc,nmax,fortran_conventions,pd,pform,pechar)
char **pc;
int nmax;
int fortran_conventions;
decimal_record *pd;
enum decimal_string_form *pform;
char **pechar;

void file_to_decimal(pc,nmax,fortran_conventions,pd,pform,pechar,pf,pnread)
char **pc;
int nmax;
int fortran_conventions;
decimal_record *pd;
enum decimal_string_form *pform;
char **pechar;
FILE *pf;
int *pnread;

void func_to_decimal(pc,nmax,fortran_conventions,pd,pform,pechar,pget,pnread,punget)
char **pc;
int nmax;
int fortran_conventions;
decimal_record *pd;
enum decimal_string_form *pform;
char **pechar;
int (*pget)();
int *pnread;
int (*punget)();
```

DESCRIPTION

The **char_to_decimal** functions parse a numeric token from at most *nmax* characters in a string ***pc* or file **pf* or function **(*pget)()** into a decimal record **pd*, classifying the form of the string in **pform* and **pechar*. The accepted syntax is intended to be sufficiently flexible to accomodate many languages:

*whitespace value*

or

*whitespace sign value*

where *whitespace* is any number of characters defined by *isspace* in **/usr/include/ctype.h**, *sign* is either of [+−], and *value* can be *number*, *nan*, or *inf*. *inf* can be INF (*inf_form*) or INFINITY (*infinity_form*) without regard to case. *nan* can be NAN (*nan_form*) or NAN(nstring) (*nanstring_form*) without regard to case; nstring is any string of characters not containing ')' or NULL; nstring is copied to pd->ds and, currently, not used subsequently. *number* consists of

*significand*

or

*significand efield*

where *significand* must contain one or more digits and may contain one point; possible forms are

| | |
|---|---|
| *digits* | *(int_form)* |
| *digits.* | *(intdot_form)* |
| *.digits* | *(dotfrac_form)* |
| *digits.digits* | *(intdotfrac_form)* |

*efield* consists of

*echar digits*

or

*echar sign digits*

where *echar* is one of [Ee], and *digits* contains one or more digits.

When *fortran_conventions* is nonzero, additional input forms are accepted according to various Fortran conventions:

0 no Fortran conventions
1 Fortran list-directed input conventions
2 Fortran formatted input conventions, ignore blanks (BN)
3 Fortran formatted input conventions, blanks are zeros (BZ)

When *fortran_conventions* is nonzero, *echar* may also be one of [Dd], and *efield* may also have the form

*sign digits*

When *fortran_conventions*>= 2, blanks may appear in the *digits* strings for the integer, fraction, and exponent fields and may appear between *echar* and the exponent sign and after the infinity and NaN forms. If *fortran_conventions*== 2, the blanks are ignored. When *fortran_conventions*== 3, the blanks that appear in *digits* strings are interpreted as zeros, and other blanks are ignored.

The form of the accepted decimal string is placed in **peform*. If an *efield* is recognized, **pechar* is set to point to the *echar*.

On input, **pc* points to the beginning of a character string buffer of length >= *nmax*. On output, **pc* points to a character in that buffer, one past the last accepted character. **string_to_decimal()** gets its characters from the buffer; **file_to_decimal()** gets its characters from **pf* and records them in the buffer, and places a null after the last character read. **func_to_decimal()** gets its characters from an int function *(*pget)()*.

The scan continues until no more characters could possibly fit the acceptable syntax or until *nmax* characters have been scanned. If the *nmax* limit is not reached then at least one extra character will usually be scanned that is not part of the accepted syntax. **file_to_decimal()** and **func_to_decimal()** set **pnread* to the number of characters read from the file; if greater than *nmax*, some characters were lost. If no characters were lost, **file_to_decimal()** and **func_to_decimal()** attempt to push back, with **ungetc**(3S) or **(*punget)()**, as many as possible of the excess characters read, adjusting **pnread* accordingly. If all unget calls are successful, then ***pc* will be NULL. No push back will be attempted if **(*punget)()** is NULL.

Typical declarations for ***pget()** and ***punget()** are:

```
int xget()
{ ... }
int (*pget)() = xget ;
int xunget(c)
char c ;
{ ... }
int (*punget)() = xunget ;
```

If no valid number was detected, *pd->fpclass* is set to **fp_signaling**, **pc* is unchanged, and **pform* is set to **invalid_form**.

**atof** and **strtod**(3) use **string_to_decimal**. **scanf**(3S) uses **file_to_decimal**.

FILES

**/usr/include/ctype.h**

SEE ALSO

**scanf**(3S), **strtod**(3), **ungetc**(3S)

NAME

strtod, atof – convert string to double-precision number

SYNOPSIS

**double strtod(str, ptr)**
**char *str, **ptr;**

**double atof(str)**
**char *str;**

DESCRIPTION

**strtod()** returns as a double-precision floating-point number the value represented by the character string pointed to by *str*. The string is scanned up to the first unrecognized character, using **string_to_decimal**(3), with *fortran_conventions* set to 0.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no number can be formed, **ptr* is set to *str*, and for historical compatibility, 0.0 is returned, although a NaN would better match the IEEE Floating-Point Standard's intent.

**atof(str)** is equivalent to **strtod(str, (char **)NULL)**. Thus, when **atof(str)** returns 0.0 there is no way to determine whether *str* contained a valid numerical string representing 0.0 or an invalid numerical string.

SEE ALSO

**scanf**(3S), **string_to_decimal**(3)

DIAGNOSTICS

Exponent overflow and underflow produce the results specified by the IEEE Standard. In addition, **errno** is set to ERANGE.

NAME

strtol, atol, atoi – convert string to integer

SYNOPSIS

**long strtol(str, ptr, base)**
**char *str, **ptr;**
**int base;**

**long atol(str)**
**char *str;**

**int atoi(str)**
**char *str;**

DESCRIPTION

**strtol()** returns as a long integer the value represented by the character string pointed to by *str*. The string is scanned up to the first character inconsistent with the base. Leading "white-space" characters (as defined by **isspace()** in **ctype**(3)) are ignored.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no integer can be formed, that location is set to *str*, and zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored, and "0x" or "0X" is ignored if *base* is 16.

If *base* is zero, the string itself determines the base thusly: after an optional leading sign a leading zero indicates octal conversion, and a leading "0x" or "0X" hexadecimal conversion. Otherwise, decimal conversion is used.

Truncation from long to int can, of course, take place upon assignment or by an explicit cast.

**atol**(*str*) is equivalent to **strtol**(*str*, (*char* **)NULL, 10).

**atoi**(*str*) is equivalent to (int) **strtol**(str, (char **)NULL, 10).

SEE ALSO

**ctype**(3), **scanf**(3S), **strtod**(3)

BUGS

Overflow conditions are ignored.

NAME

stty, gtty – set and get terminal state

SYNOPSIS

```
#include <sgtty.h>

stty(fd, buf)
int fd;
struct sgttyb *buf;

gtty(fd, buf)
int fd;
struct sgttyb *buf;
```

DESCRIPTION

**This interface is obsoleted by ioctl**(2).

**stty**( ) sets the state of the terminal associated with *fd*. **stty**( ) retrieves the state of the terminal associated with *fd*. To set the state of a terminal the call must have write permission.

The **stty**( ) call is actually

**ioctl(fd, TIOCSETP, buf)**

while the **stty**( ) call is

**ioctl(fd, TIOCGETP, buf)**

See **ioctl**(2) and **ttcompat**(4M) for an explanation.

DIAGNOSTICS

If the call is successful 0 is returned, otherwise −1 is returned and the global variable **errno** contains the reason for the failure.

SEE ALSO

**ioctl**(2), **ttcompat**(4M)

NAME

swab – swap bytes

SYNOPSIS

**swab(from, to, nbytes)**
**char *from, *to;**

DESCRIPTION

**swab( )** copies *nbytes* bytes pointed to by **from** to the position pointed to by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between high-ender machines (IBM 360's, MC68000's, etc) and low-endian machines (such as Sun386i).

*nbytes* should be even.

The *from* and *to* addresses should not overlap in portable programs.

## NAME

syslog, openlog, closelog, setlogmask – control system log

## SYNOPSIS

**#include <syslog.h>**

**openlog(ident, logopt, facility)**
**char *ident;**

**syslog(priority, message, parameters ... )**
**char *message;**

**closelog( )**

**setlogmask(maskpri)**

## DESCRIPTION

**syslog( )** passes *message* to **syslogd**(8), which logs it in an appropriate system log, writes it to the system console, forwards it to a list of users, or forwards it to the **syslogd** on another host over the network. The message is tagged with a priority of *priority*. The message looks like a **printf**(3S) string except that **%m** is replaced by the current error message (collected from **errno**). A trailing NEWLINE is added if needed.

Priorities are encoded as a *facility* and a *level*. The facility describes the part of the system generating the message. The level is selected from an ordered list:

| | |
|---|---|
| LOG_EMERG | A panic condition. This is normally broadcast to all users. |
| LOG_ALERT | A condition that should be corrected immediately, such as a corrupted system database. |
| LOG_CRIT | Critical conditions, such as hard device errors. |
| LOG_ERR | Errors. |
| LOG_WARNING | Warning messages. |
| LOG_NOTICE | Conditions that are not error conditions, but that may require special handling. |
| LOG_INFO | Informational messages. |
| LOG_DEBUG | Messages that contain information normally of use only when debugging a program. |

If special processing is needed, **openlog( )** can be called to initialize the log file. The parameter *ident* is a string that is prepended to every message. *logopt* is a bit field indicating logging options. Current values for *logopt* are:

| | |
|---|---|
| LOG_PID | Log the process ID with each message. This is useful for identifying specific daemon processes (for daemons that fork). |
| LOG_CONS | Write messages to the system console if they cannot be sent to **syslogd**. This option is safe to use in daemon processes that have no controlling terminal, since **syslog( )** forks before opening the console. |
| LOG_NDELAY | Open the connection to **syslogd** immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated. |
| LOG_NOWAIT | Do not wait for child processes that have been forked to log messages onto the console. This option should be used by processes that enable notification of child termination using **SIGCHLD**, since **syslog( )** may otherwise block waiting for a child whose exit status has already been collected. |

The *facility* parameter encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded:

| | |
|---|---|
| **LOG_KERN** | Messages generated by the kernel. These cannot be generated by any user processes. |
| **LOG_USER** | Messages generated by random user processes. This is the default facility identifier if none is specified. |
| **LOG_MAIL** | The mail system. |
| **LOG_DAEMON** | System daemons, such as **ftpd**(8C), **routed**(8C), etc. |
| **LOG_AUTH** | The authorization system: **login**(1), **su**(1), **getty**(8), etc. |
| **LOG_LPR** | The line printer spooling system: **lpr**(1), **lpc**(8), **lpd**(8), etc. |
| **LOG_NEWS** | Reserved for the USENET network news system. |
| **LOG_UUCP** | Reserved for the UUCP system; it does not currently use **syslog**. |
| **LOG_CRON** | The **cron/at** facility; **crontab**(1), **at**(1), **cron**(8), etc. |
| **LOG_LOCAL0–7** | Reserved for local use. |

**closelog()** can be used to close the log file.

**setlogmask()** sets the log priority mask to *maskpri* and returns the previous mask. Calls to **syslog()** with a priority not set in *maskpri* are rejected. The mask for an individual priority *pri* is calculated by the macro **LOG_MASK**(*pri*); the mask for all priorities up to and including *toppri* is given by the macro **LOG_UPTO**(*toppri*). The default allows all priorities to be logged.

## EXAMPLES

This call logs a message at priority **LOG_ALERT**:

```
syslog(LOG_ALERT, "who: internal error 23");
```

The FTP daemon **ftpd** would make this call to **openlog()** to indicate that all messages it logs should have an identifying string of **ftpd**, should be treated by **syslogd** as other messages from system daemons are, should include the process ID of the process logging the message:

```
openlog("ftpd", LOG_PID, LOG_DAEMON);
```

Then it would make the following call to **setlogmask()** to indicate that messages at priorities from **LOG_EMERG** through **LOG_ERR** should be logged, but that no messages at any other priority should be logged:

```
setlogmask(LOG_UPTO(LOG_ERR));
```

Then, to log a message at priority **LOG_INFO**, it would make the following call to **syslog**:

```
syslog(LOG_INFO, "Connection from host %d", CallingHost);
```

A locally-written utility could use the following call to **syslog()** to log a message at priority **LOG_INFO** to be treated by **syslogd** as other messages to the facility **LOG_LOCAL2** are:

```
syslog(LOG_INFO|LOG_LOCAL2, "error: %m");
```

## SEE ALSO

**at**(1), **crontab**(1), **logger**(1), **login**(1), **lpr**(1), **su**(1), **printf**(3S), **syslog.conf**(5), **cron**(8), **ftpd**(8C), **getty**(8), **lpc**(8), **lpd**(8), **routed**(8C), **syslogd**(8)

NAME

system – issue a shell command

SYNOPSIS

**system(string)**
**char *string;**

DESCRIPTION

**system( )** gives the *string* to **sh**(1) as input, just as if the string had been typed as a command from a terminal. The current process performs a **wait**(2) system call, and waits until the shell terminates. **system( )** then returns the exit status returned by **wait**. Unless the the shell was interrupted by a signal, its termination status is contained in the 8 bits higher up from the low-order 8 bits of the value returned by **wait**.

SEE ALSO

**sh**(1), **execve**(2), **wait**(2), **popen**(3S)

DIAGNOSTICS

Exit status 127 (may be displayed as "32512") indicates the shell could not be executed.

## NAME

termcap, tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – terminal independent operation routines

## SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum (id)
char *id;

tgetflag (id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

## DESCRIPTION

These functions extract and use capabilities from the terminal capability data base **termcap**(5). These are low level routines; see **curses**(3X) for a higher level package.

**tgetent()** extracts the entry for terminal *name* into the *bp* buffer, with the current size of the tty (usually a window). This allows pre-SunWindows programs to run in a window of arbitrary size. *bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to **tgetnum**, **tgetflag**, and **tgetstr**. **tgetent()** returns –1 if it cannot open the **termcap()** file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type *name* is the same as the environment string TERM, the TERMCAP string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than **/etc/termcap**. This can speed up entry into programs that call **tgetent**, as well as to help debug new terminal descriptions or to make one for your terminal if you cannot write the file **/etc/termcap**. Note: if the window size changes, the "lines" and "columns" entries in *bp* are no longer correct. See the *SunView 1 Programmer's Guide* for details regarding [how to handle] this.

**tgetnum()** gets the numeric value of capability ID, returning –1 if is not given for the terminal. **tgetflag()** returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. **tgetstr()** gets the string value of capability ID, placing it in the buffer at *area*, advancing the *area* pointer. It decodes the abbreviations for this field described in **termcap**(5), except for cursor addressing and padding information. **tgetstr()** returns the string pointer if successful. Otherwise it returns zero.

**tgoto( )** returns a cursor addressing string decoded from *cm* to go to column *destcol* in line *destline*. It uses the external variables **UP** (from the **up** capability) and **BC** (if **bc** is given rather than **bs**) if necessary to avoid placing **\n**, **^D** or **^@** in the returned string. (Programs which call **tgoto( )** should be sure to turn off the **XTABS bit**(s),**since tgoto( )** may now output a tab. Note: programs using **termcap( )** should in general turn off **XTABS** anyway since some terminals use **^I** (CTRL-I) for other functions, such as nondestructive space.) If a **%** sequence is given which is not understood, then **tgoto( )** returns **OOPS**.

**tputs( )** decodes the leading padding information of the string *cp*; *affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable, *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the encoded output speed of the terminal as described in **tty**(4). The external variable **PC** should contain a pad character to be used (from the **pc** capability) if a NULL (**^@**) is inappropriate.

FILES

| | |
|---|---|
| **/usr/lib/libtermcap.a** | –ltermcap library |
| **/etc/termcap** | data base |

SEE ALSO

**ex**(1), **curses**(3X), **tty**(4), **termcap**(5)

**NAME**

time, ftime – get date and time

**SYNOPSIS**

```
#include <sys/types.h>
#include <sys/timeb.h>

time_t timeofday = time((time_t *)0)
time_t timeofday = time(tloc)
time_t *tloc;

ftime(tp)
struct timeb *tp;
```

**DESCRIPTION**

**time()** returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If **tloc** is non-NULL, the return value is also stored in the place to which **tloc** points.

The **ftime()** entry fills in a structure pointed to by its argument, as defined by **<sys/timeb.h>**:

```
struct timeb
{
        time_t    time;
        unsigned short millitm;
        short     timezone;
        short     dstflag;
};
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

**SEE ALSO**

**date**(1V), **gettimeofday**(2), **ctime**(3)

NAME

times – get process times

SYNOPSIS

```
#include <sys/types.h>
#include <sys/times.h>

times(buffer)
struct tms *buffer;
```

DESCRIPTION

**This interface is obsoleted by getrusage(2).**

**times**() returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ is 60.

This is the structure returned by **times**:

```
struct tms {
        time_t  tms_utime;          /* user time */
        time_t  tms_stime;          /* system time */
        time_t  tms_cutime;         /* user time, children */
        time_t  tms_cstime;         /* system time, children */
};
```

The children's times are the sum of the children's process times and their children's times.

SEE ALSO

**time**(1V), **getrusage**(2), **wait**(2), **time**(3C)

## NAME

timezone – get time zone name given offset from GMT

## SYNOPSIS

**char *timezone(zone, dst)**

## DESCRIPTION

**timezone( )** attempts to return the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Savings Time version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; for instance, in Afghanistan **'timezone(–(60*4+30), 0)'** is appropriate because it is 4:30 ahead of GMT and the string **GMT+4:30** is produced.

Note: the offset westward from Greenwich and an indication of whether Daylight Savings Time is in effect may not be sufficient to determine the name of the time zone, as the name may differ between different locations in the same time zone. Instead of using **timezone( )** to determine the name of the time zone for a given time, that time should be converted to a ' **struct tm**' using **localtime** (see **ctime**(3)) and the *tm_zone* field of that structure should be used. **timezone( )** is retained for compatibility with existing programs.

## SEE ALSO

**ctime**(3)

NAME

tmpfile – create a temporary file

SYNOPSIS

**#include <stdio.h>**

**FILE *tmpfile( )**

DESCRIPTION

**tmpfile( )** creates a temporary file using a name generated by **tmpnam**(3S), and returns a corresponding **FILE** pointer. If the file cannot be opened, an error message is printed using **perror**(3), and a NULL pointer is returned. The file will automatically be deleted when the process using it terminates. The file is opened for update (''w+'').

SEE ALSO

**creat**(2), **unlink**(2), **fopen**(3S), **mktemp**(3), **perror**(3), **tmpnam**(3S)

NAME

tmpnam, tempnam – create a name for a temporary file

SYNOPSIS

**#include <stdio.h>**

**char *tmpnam (s)**
**char *s;**

**char *tempnam (dir, pfx)**
**char *dir, *pfx;**

DESCRIPTION

These functions generate file names that can safely be used for a temporary file.

**tmpnam()** always generates a file name using the path-prefix defined as **P_tmpdir** in the **<stdio.h>** header file. If *s* is NULL, **tmpnam()** leaves its result in an internal static area and returns a pointer to that area. The next call to **tmpnam()** will destroy the contents of the area. If *s* is not NULL, it is assumed to be the address of an array of at least **L_tmpnam** bytes, where **L_tmpnam** is a constant defined in **<stdio.h>**; **tmpnam()** places its result in that array and returns *s*.

**tempnam()** allows the user to control the choice of a directory. The argument **dir** points to the name of the directory in which the file is to be created. If **dir** is NULL or points to a string which is not a name for an appropriate directory, the path-prefix defined as **P_tmpdir** in the **<stdio.h>** header file is used. If that directory is not accessible, **/tmp** will be used as a last resort. This entire sequence can be up-staged by providing an environment variable **TMPDIR** in the user's environment, whose value is the name of the desired temporary-file directory.

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the *pfx* argument for this. This argument may be NULL or point to a string of up to five characters to be used as the first few characters of the temporary-file name.

**tempnam()** uses **malloc** to get space for the constructed file name, and returns a pointer to this area. Thus, any pointer value returned from **tempnam()** may serve as an argument to *free* (see **malloc**(3)). If **tempnam()** cannot return the expected result for any reason, that is, **malloc** failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer will be returned.

NOTES

These functions generate a different file name each time they are called.

Files created using these functions and either *fopen* or *creat* are temporary only in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use **unlink**(2) to remove the file when its use is ended.

SEE ALSO

**creat**(2), **unlink**(2), **fopen**(3S), **malloc**(3), **mktemp**(3), **tmpfile**(3S)

BUGS

If called more than 17,576 times in a single process, these functions will start recycling previously used names.

Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using these functions or *mktemp*, and the file names are chosen so as to render duplication by other means unlikely.

NAME

tsearch, tfind, tdelete, twalk – manage binary search trees

SYNOPSIS

**#include <search.h>**

**char *tsearch ((char *) key, (char **) rootp, compar)**
**int (*compar)( );**

**char *tfind ((char *) key, (char **) rootp, compar)**
**int (*compar)( );**

**char *tdelete ((char *) key, (char **) rootp, compar)**
**int (*compar)( );**

**void twalk ((char *) root, action)**
**void (*action)( );**

DESCRIPTION

**tsearch, tfind, tdelete,** and **twalk()** are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

**tsearch()** is used to build and access the tree. *key* is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to **key* (the value pointed to by *key*), a pointer to this found datum is returned. Otherwise, **key* is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. *rootp* points to a variable that points to the root of the tree. A NULL value for the variable pointed to by *rootp* denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.

Like **tsearch, tfind()** will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, **tfind()** will return a NULL pointer. The arguments for **tfind()** are the same as for **tsearch.**

**tdelete()** deletes a node from a binary search tree. The arguments are the same as for **tsearch.** The variable pointed to by *rootp* will be changed if the deleted node was the root of the tree. **tdelete()** returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

**twalk()** traverses a binary search tree. *root* is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type **typedef enum { preorder, postorder, endorder, leaf }** VISIT; (defined in the **<search.h>** header file), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLE

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <stdio.h>
struct node {           /* pointers to these are stored in the tree */
        char *string;
        int length;
};
char string_space[10000];       /* space to store strings */
struct node nodes[500];         /* nodes to store */
struct node *root = NULL;       /* this points to the root */
main( )
{
        char *strptr = string_space;
        struct node *nodeptr = nodes;
        void print_node( ), twalk( );
        int i = 0, node_compare( );
        while (gets(strptr) != NULL && i++ < 500)  {
                /* set node */
                nodeptr->string = strptr;
                nodeptr->length = strlen(strptr);
                /* put node into the tree */
                 (void) tsearch((char *)nodeptr, &root,
                        node_compare);
                /* adjust pointers, so we don't overwrite tree */
                strptr += nodeptr->length + 1;
                nodeptr++;
        }
        twalk(root, print_node);
}
/*
        This routine compares two nodes, based on an
        alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
struct node *node1, *node2;
{
        return strcmp(node1->string, node2->string);
}
/*
        This routine prints out a node, the first time
        twalk encounters it.
*/
void
print_node(node, order, level)
struct node **node;
VISIT order;
int level;
{
        if (order == preorder || order == leaf)  {
                (void)printf("string = %20s,  length = %d\n",
                        ( (**node)->string, (*node)->length);
        }
}
```

**SEE ALSO**

**bsearch**(3), **hsearch**(3), **lsearch**(3)

**DIAGNOSTICS**

A NULL pointer is returned by **tsearch()** if there is not enough space available to create a new node.

A NULL pointer is returned by **tsearch, tfind()** and **tdelete()** if *rootp* is NULL on entry.

If the datum is found, both **tsearch()** and **tfind()** return a pointer to it. If not, **tfind()** returns NULL, and **tsearch()** returns a pointer to the inserted item.

**WARNINGS**

The *root* argument to **twalk()** is one level of indirection less than the *rootp* arguments to **tsearch()** and **tdelete.**

There are two nomenclatures used to refer to the order in which tree nodes are visited. **tsearch()** uses preorder, postorder and endorder to respectively refer to visting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

**BUGS**

If the calling function alters the pointer to the root, results are unpredictable.

NAME

ttyname, isatty – find name of a terminal

SYNOPSIS

**char *ttyname(filedes)**

**isatty(filedes)**

DESCRIPTION

**ttyname()** returns a pointer to the NULL-terminated path name of the terminal device associated with file descriptor *filedes*.

**isatty()** returns 1 if *filedes* is associated with a terminal device, 0 otherwise.

FILES

**/dev/***

SEE ALSO

**ioctl**(2), **ttytab**(5)

DIAGNOSTICS

**ttyname()** returns a NULL pointer if *filedes* does not describe a terminal device in directory **/dev**.

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

ttyslot – find the slot in the utmp file of the current process

SYNOPSIS

**ttyslot()**

DESCRIPTION

**ttyslot()** returns the index of the current user's entry in the **/etc/utmp** file. This is accomplished by actually scanning the file **/etc/ttys** for the name of the terminal associated with the standard input, the standard output, or the error output (0, 1 or 2).

FILES

**/etc/ttys**
**/etc/utmp**

DIAGNOSTICS

A value of 0 is returned if an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

NAME

ualarm – schedule signal after interval in microseconds

SYNOPSIS

**unsigned ualarm(value, interval)**
**unsigned value;**
**unsigned interval;**

DESCRIPTION

**This is a simplified interface to setitimer** (see **getitimer**(2)).

**ualarm()** sends signal **SIGALRM**, see **signal**(3), to the invoking process in a number of microseconds given by the *value* argument. Unless caught or ignored, the signal terminates the process.

If the *interval* argument is non-zero, the **SIGALRM** signal will be sent to the process every *interval* microseconds after the timer expires (for instance, after *value* microseconds have passed).

Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 2147483647 microseconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

**getitimer**(2), **sigpause**(2), **sigvec**(2), **alarm**(3C), **signal**(3), **sleep**(3), **usleep**(3)

NAME

ulimit – get and set user limits

SYNOPSIS

**long ulimit(cmd, newlimit)**
**int cmd;**
**long newlimit;**

DESCRIPTION

This function is included for System V compatibility.

This routine provides for control over process limits. The **cmd** values available are:

1 Get the process's file size limit. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.

2 Set the process's file size limit to the value of *newlimit*. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. **ulimit**() will fail and the limit will be unchanged if a process with an effective user ID other than the super-user attempts to increase its file size limit.

3 Get the maximum possible break value. See **brk**(2).

4 Get the size of the process' file descriptor table, as returned by **getdtablesize**(2).

RETURN VALUE

Upon successful completion, a non-negative value is returned. Otherwise a value of −1 is returned and **errno** is set to indicate the error.

ERRORS

The following error codes may be set in **errno**:

EPERM A user other than the super-user attempted to increase the file size limit.

SEE ALSO

**brk**(2), **getdtablesize**(2), **getrlimit**(2), **write**(2V)

NAME

ungetc – push character back into input stream

SYNOPSIS

**#include <stdio.h>**

**ungetc(c, stream)**
**FILE *stream;**

DESCRIPTION

**ungetc()** pushes the character *c* back onto an input stream. That character will be returned by the next **getc()** call on that stream. **ungetc()** returns *c*, and leaves the file stream unchanged.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. In the case that stream is **stdin**, one character may be pushed back onto the buffer without a previous read statement.

If *c* equals EOF, **ungetc()** does nothing to the buffer and returns EOF.

An **fseek**(3S) erases all memory of pushed back characters.

SEE ALSO

**fseek**(3S), **getc**(3S), **setbuf**(3S)

DIAGNOSTICS

**ungetc()** returns EOF if it cannot push a character back.

NAME

usleep – suspend execution for interval in microseconds

SYNOPSIS

**usleep(useconds)**
**unsigned useconds;**

DESCRIPTION

Suspend the current process for the number of microseconds specified by the argument. The actual suspension time may be an arbitrary amount longer because of other activity in the system, or because of the time spent in processing the call.

The routine is implemented by setting an interval timer and pausing until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent a short time later.

This routine is implemented using **setitimer** (see **getitimer**(2)); it requires eight system calls each time it is invoked. A similar but less compatible function can be obtained with a single **select**(2); it would not restart after signals, but would not interfere with other uses of **setitimer**.

SEE ALSO

**getitimer**(2), **sigpause**(2), **alarm**(3C), **sleep**(3), **ualarm**(3)

NAME

utime – set file times

SYNOPSIS

**#include <sys/types.h>**

**int utime(file, timep)**
**char *file;**
**time_t *timep;**

DESCRIPTION

**utime()** sets the access and modification times of the file named by *file*.

If the *timep* argument is NULL, the access and modification times are set to the current time. A process must be the owner of the file or have write permission for the file to use **utime()** in this manner.

If the *timep* argument is not NULL, it is assumed to point to an array of two **time_t** values. The access time is set to the value of the first member, and the modification time is set to the value of the second member. The times contained in that array are measured in seconds since 00:00:00 GMT Jan 1, 1970. Only the owner of the file or the super-user may use **utime()** in this manner.

In either case, the "inode-changed" time of the file is set to the current time.

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and **errno** is set to indicate the error.

ERRORS

**utime()** will fail if one or more of the following are true:

| | |
|---|---|
| ENOTDIR | A component of the path prefix of *file* is not a directory. |
| ENAMETOOLONG | The length of a component of *file* exceeds 255 characters, or the length of *file* exceeds 1023 characters. |
| ENOENT | The file referred to by *file* does not exist. |
| EACCESS | Search permission is denied for a component of the path prefix of *file*. |
| ELOOP | Too many symbolic links were encountered in translating *file*. |
| EPERM | The effective user ID of the process is not super-user and not the owner of the file, and *timep* is not NULL. |
| EACCESS | The effective user ID is not super-user and not the owner of the file, write permission is denied for the file, and *timep* is NULL. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| EROFS | The file system containing the file is mounted read-only. |
| EFAULT | *file* or *timep* points outside the process's allocated address space. |

SEE ALSO

**stat**(2), **utimes**(2)

NAME

values – machine-dependent values

SYNOPSIS

**#include <values.h>**

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

| | |
|---|---|
| BITS(*type*) | The number of bits in a specified type (for instance, int). |
| HIBITS | The value of a short integer with only the high-order bit set (in most implementations, 0x8000). |
| HIBITL | The value of a long integer with only the high-order bit set (in most implementations, 0x80000000). |
| HIBITI | The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL). |
| MAXSHORT | The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767). |
| MAXLONG | The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647). |
| MAXINT | The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG). |
| MAXFLOAT | |
| LN_MAXFLOAT | The maximum value of a single-precision floating-point number, and its natural logarithm. |
| MAXDOUBLE | |
| LN_MAXDOUBLE | The maximum value of a double-precision floating-point number, and its natural logarithm. |
| MINFLOAT | |
| LN_MINFLOAT | The minimum positive value of a single-precision floating-point number, and its natural logarithm. |
| MINDOUBLE | |
| LN_MINDOUBLE | The minimum positive value of a double-precision floating-point number, and its natural logarithm. |
| FSIGNIF | The number of significant bits in the mantissa of a single-precision floating-point number. |
| DSIGNIF | The number of significant bits in the mantissa of a double-precision floating-point number. |

FILES

**/usr/include/values.h**

SEE ALSO

**intro**(3), **intro**(3M)

NAME

varargs – handle variable argument list

SYNOPSIS

**#include <varargs.h>**

**function(va_alist) va_dcl**

**va_list pvar ;**

**va_start**

**f = va_arg(pvar, type);**

**va_end(pvar);**

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as **printf**(3S)) but do not use **varargs**( ) are inherently nonportable, since different machines use different argument passing conventions. Routines with variable arguments lists *must* use **varargs**( ) functions in order to run correctly on Sun-4 systems.

**va_alist** is used in a function header to declare a variable argument list.

**va_dcl** is a declaration for **va_alist**. No semicolon should follow **va_dcl**.

**va_list** is a type defined for the variable used to traverse the list. One such variable must always be declared.

**va_start**(*pvar*) is called to initialize *pvar* to the beginning of the list.

**va_arg**(*pvar*, *type*) will return the next argument in the list pointed to by *pvar*. The parameter *type* is a type name such that the type of a pointer to an object that has the specified type can be obtained simply by appending a * to *type*. If *type* disagrees with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined.

In standard C, arguments that are **char** or **short** are converted to **int** and should be accessed as **int**, arguments that are **unsigned char** or **unsigned short** are converted to **unsigned int** and should be accessed as **unsigned int**, and arguments that are **float** are converted to **double** and should be accessed as **double**. Different types can be mixed, but it is up to the routine to know what type of argument is expected, since it cannot be determined at runtime.

**va_end**(*pvar*) is used to finish up.

Multiple traversals, each bracketed by **va_start** ... **va_end**, are possible.

**va_alist** must encompass the entire arguments list. This insures that a **#define** statement can be used to redefine or expand its value.

The argument list (or its remainder) can be passed to another function using a pointer to a variable of type **va_list**— in which case a call to **va_arg** in the subroutine advances the argument-list pointer with respect to the caller as well.

EXAMPLE

This example is a possible implementation of **execl**(3).

```
#include <varargs.h>
#define MAXARGS        100

/*      execl is called by
*       execl(file, arg1, arg2, ..., (char *)0);
*/
execl (va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[MAXARGS];
        int argno = 0;

        va_start (ap);
        file = va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
                ;
        va_end (ap);
        return execv(file, args);
}
```

SEE ALSO

**execl**(3), **printf**(3S)

BUGS

It is up to the calling routine to specify how many arguments there are, since it is not possible to determine this from the stack frame. For example, **execl**() is passed a zero pointer to signal the end of the list. **printf**() can tell how many arguments are supposed to be there by the format.

The macros **va_start** and **va_end** may be arbitrarily complex; for example, **va_start** might contain an opening brace, which is closed by a matching brace in **va_end**. Thus, they should only be used where they could be placed within a single complex statement.

NAME

vlimit – control maximum system resource consumption

SYNOPSIS

**#include <sys/vlimit.h>**

**vlimit(resource, value) int resource, value;**

DESCRIPTION

**This facility is superseded by getrlimit(2).**

Limits the consumption by the current process and each process it creates to not individually exceed *value* on the specified *resource*. If *value* is specified as –1, then the current limit is returned and the limit is unchanged. The resources which are currently controllable are:

| | |
|---|---|
| **LIM_NORAISE** | A pseudo-limit; if set non-zero then the limits may not be raised. Only the super-user may remove the *noraise* restriction. |
| **LIM_CPU** | the maximum number of CPU-seconds to be used by each process |
| **LIM_FSIZE** | the largest single file which can be created |
| **LIM_DATA** | the maximum growth of the data+stack region using **sbrk** (see **brk**(2)) beyond the end of the program text |
| **LIM_STACK** | the maximum size of the automatically-extended stack region |
| **LIM_CORE** | the size of the largest core dump that will be created. |
| **LIM_MAXRSS** | a soft limit for the amount of physical memory (in bytes) to be given to the program. If memory is tight, the system will prefer to take memory from processes which are exceeding their declared **LIM_MAXRSS.** |

Because this information is stored in the per-process information this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to **csh**(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way; a *break* call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file I/O operation which would create a file which is too large will cause a signal **SIGXFSZ** to be generated, this normally terminates the process, but may be caught. When the cpu time limit is exceeded, a signal **SIGXCPU** is sent to the offending process; to allow it time to process the signal it is given 5 seconds grace by raising the CPU time limit.

SEE ALSO

**csh**(1), **sh**(1), **brk**(2)

BUGS

If **LIM_NORAISE** is set, then no grace should be given when the CPU time limit is exceeded.

There should be *limit* and *unlimit* commands in **sh**(1) as well as in **csh**(1).

NAME

vprintf, vfprintf, vsprintf – print formatted output of a varargs argument list

SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf(format, ap)
char *format;
va_list ap;

int vfprintf(stream, format, ap)
FILE *stream;
char *format;
va_list ap;

char *vsprintf(s, format, ap)
char *s, *format;
va_list ap;
```

DESCRIPTION

**vprintf**, **vfprintf**, and **vsprintf**( ) are the same as **printf**(3S), **fprintf**, and **sprintf** respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **varargs**(3).

EXAMPLE

The following demonstrates how **vfprintf**( ) could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
...
        /*  error should be called like:
        *       error(function_name, format, arg1, arg2...);
        *  Note: function_name and format cannot be declared
        *  separately because of the definition of varargs.
        */

/*VARARGS0*/
void
error (va_alist)
        va_dcl;
{
        va_list args;
        char *fmt;

        va_start(args);
                /* print name of function causing error */
        (void) fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
        fmt = va_arg(args, char *);
                /* print out remainder of message */
        (void) vfprintf(stderr, fmt, args);
        va_end(args);
        (void) abort( );
}
```

SEE ALSO

**printf**(3S), **varargs**(3)

NAME

vsyslog – log message with a varargs argument list

SYNOPSIS

```
#include <syslog.h>
#include <varargs.h>

int vsyslog(priority, message, ap)
char *message;
va_list ap;
```

DESCRIPTION

vsyslog( ) is the same as syslog(3) except that instead of being called with a variable number of arguments, it is called with an argument list as defined by varargs(3).

EXAMPLE

The following demonstrates how vsyslog( ) could be used to write an error routine.

```
#include <syslog.h>
#include <varargs.h>
...
        /*  error should be called like:
        *       error(pri, function_name, format, arg1, arg2...);
        *  Note that pri, function_name, and format cannot be declared
        *  separately because of the definition of varargs.
        */

/*VARARGS0*/
void
error(va_alist)
        va_dcl;
{
        va_list args;
        int pri;
        char *message;

        va_start(args);
        pri = va_arg(args, int);
                /* log name of function causing error */
        (void) syslog(pri, "ERROR in %s", va_arg(args, char *));
        message = va_arg(args, char *);
                /* log remainder of message */
        (void) vsyslog(pri, fmt, args);
        va_end(args);
        (void) abort( );
}
```

SEE ALSO

syslog(3), varargs(3)

NAME

vtimes – get information about resource utilization

SYNOPSIS

**vtimes(par_vm, ch_vm)**
**struct vtimes *par_vm, *ch_vm;**

DESCRIPTION

**This facility is superseded by getrusage(2).**

**vtimes()** returns accounting information for the current process and for the terminated child processes of the current process. Either *par_vm* or *ch_vm* or both may be 0, in which case only the information for the pointers which are non-zero is returned.

After the call, each buffer contains information as defined by the contents of the include file **<sys/vtimes.h>**:

```
struct vtimes {
        int     vm_utime;               /* user time (*HZ) */
        int     vm_stime;               /* system time (*HZ) */
        /* divide next two by utime+stime to get averages */
        unsigned vm_idsrss;             /* integral of d+s rss */
        unsigned vm_ixrss;              /* integral of text rss */
        int     vm_maxrss;              /* maximum rss */
        int     vm_majflt;              /* major page faults */
        int     vm_minflt;              /* minor page faults */
        int     vm_nswap;               /* number of swaps */
        int     vm_inblk;               /* block reads */
        int     vm_oublk;               /* block writes */
};
```

The **vm_utime** and **vm_stime** fields give the user and system time respectively in 60ths of a second (or 50ths if that is the frequency of wall current in your locality.) The **vm_idrss** and **vm_ixrss** measure memory usage. They are computed by integrating the number of memory pages in use each over cpu time. They are reported as though computed discretely, adding the current memory usage (in 512 byte pages) each time the clock ticks. If a process used 5 core pages over 1 cpu-second for its data and stack, then **vm_idsrss** would have the value 5*60, where **vm_utime+vm_stime** would be the 60. **vm_idsrss** integrates data and stack segment usage, while **vm_ixrss** integrates text segment usage. **vm_maxrss** reports the maximum instantaneous sum of the text+data+stack core-resident page count.

The **vm_majflt** field gives the number of page faults which resulted in disk activity; the **vm_minflt** field gives the number of page faults incurred in simulation of reference bits; **vm_nswap** is the number of swaps which occurred. The number of file system input/output events are reported in **vm_inblk** and **vm_oublk** These numbers account only for real I/O; data supplied by the caching mechanism is charged only to the first process to read or write the data.

SEE ALSO

**getrusage(2)**, **wait(2)**

NAME

xdr – library routines for external data representation

SYNOPSIS AND DESCRIPTION

These routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls are transmitted using these routines.

**xdr_array(xdrs, arrp, sizep, maxsize, elsize, elproc)**
**XDR *xdrs;**
**char **arrp;**
**u_int *sizep, maxsize, elsize;**
**xdrproc_t elproc;**

A filter primitive that translates between variable-length arrays and their corresponding external representations. The parameter *arrp* is the address of the pointer to the array, while *sizep* is the address of the element count of the array; this element count cannot exceed *maxsize*. The parameter *elsize* is the *sizeof* each of the array's elements, and *elproc* is an XDR filter that translates between the array elements' C form, and their external representation. This routine returns one if it succeeds, zero otherwise.

**xdr_bool(xdrs, bp)**
**XDR *xdrs;**
**bool_t *bp;**

A filter primitive that translates between booleans (C integers) and their external representations. When encoding data, this filter produces values of either one or zero. This routine returns one if it succeeds, zero otherwise.

**xdr_bytes(xdrs, sp, sizep, maxsize)**
**XDR *xdrs;**
**char **sp;**
**u_int *sizep, maxsize;**

A filter primitive that translates between counted byte strings and their external representations. The parameter *sp* is the address of the string pointer. The length of the string is located at address *sizep*; strings cannot be longer than *maxsize*. This routine returns one if it succeeds, zero otherwise.

**xdr_char(xdrs, cp)**
**XDR *xdrs;**
**char *cp;**

A filter primitive that translates between C characters and their external representations. This routine returns one if it succeeds, zero otherwise. Note: encoded characters are not packed, and occupy 4 bytes each. For arrays of characters, it is worthwhile to consider **xdr_bytes()**, **xdr_opaque()** or **xdr_string()**.

**void**
**xdr_destroy(xdrs)**
**XDR *xdrs;**

A macro that invokes the destroy routine associated with the XDR stream, *xdrs*. Destruction usually involves freeing private data structures associated with the stream. Using *xdrs* after invoking **xdr_destroy()** is undefined.

**xdr_double(xdrs, dp)**
**XDR *xdrs;**
**double *dp;**

A filter primitive that translates between C **double** precision numbers and their external representations. This routine returns one if it succeeds, zero otherwise.

**xdr_enum(xdrs, ep)**
**XDR *xdrs;**
**enum_t *ep;**

A filter primitive that translates between C **enums** (actually integers) and their external representations. This routine returns one if it succeeds, zero otherwise.

**xdr_float(xdrs, fp)**
**XDR *xdrs;**
**float *fp;**

A filter primitive that translates between C **floats** and their external representations. This routine returns one if it succeeds, zero otherwise.

**void**
**xdr_free(proc, objp)**
**xdrproc_t proc;**
**char *objp;**

Generic freeing routine. The first argument is the XDR routine for the object being freed. The second argument is a pointer to the object itself. Note: the pointer passed to this routine is *not* freed, but what it points to *is* freed (recursively).

**u_int**
**xdr_getpos(xdrs)**
**XDR *xdrs;**

A macro that invokes the get-position routine associated with the XDR stream, *xdrs*. The routine returns an unsigned integer, which indicates the position of the XDR byte stream. A desirable feature of XDR streams is that simple arithmetic works with this number, although the XDR stream instances need not guarantee this.

**long ***
**xdr_inline(xdrs, len)**
**XDR *xdrs;**
**int len;**

A macro that invokes the in-line routine associated with the XDR stream, *xdrs*. The routine returns a pointer to a contiguous piece of the stream's buffer; *len* is the byte length of the desired buffer. Note: pointer is cast to **long ***.

Warning: **xdr_inline()** may return NULL (0) if it cannot allocate a contiguous piece of a buffer. Therefore the behavior may vary among stream instances; it exists for the sake of efficiency.

**xdr_int(xdrs, ip)**
**XDR *xdrs;**
**int *ip;**

A filter primitive that translates between C integers and their external representations. This routine returns one if it succeeds, zero otherwise.

**xdr_long(xdrs, lp)**
**XDR *xdrs;**
**long *lp;**

A filter primitive that translates between C **long** integers and their external representations. This routine returns one if it succeeds, zero otherwise.

**void**
**xdrmem_create(xdrs, addr, size, op)**
**XDR *xdrs;**
**char *addr;**
**u_int size;**
**enum xdr_op op;**

This routine initializes the XDR stream object pointed to by *xdrs*. The stream's data is written to, or read from, a chunk of memory at location *addr* whose length is no more than *size* bytes long. The *op* determines the direction of the XDR stream (either **XDR_ENCODE**, **XDR_DECODE**, or **XDR_FREE**).

**xdr_opaque(xdrs, cp, cnt)**
**XDR *xdrs;**
**char *cp;**
**u_int cnt;**

A filter primitive that translates between fixed size opaque data and its external representation. The parameter *cp* is the address of the opaque object, and *cnt* is its size in bytes. This routine returns one if it succeeds, zero otherwise.

**xdr_pointer(xdrs, objpp, objsize, xdrobj)**
**XDR *xdrs;**
**char **objpp;**
**u_int objsize;**
**xdrproc_t xdrobj;**

Like **xdr_reference()** execpt that it serializes NULL pointers, whereas **xdr_reference()** does not. Thus, **xdr_pointer()** can represent recursive data structures, such as binary trees or linked lists.

**void**
**xdrrec_create(xdrs, sendsize, recvsize, handle, readit, writeit)**
**XDR *xdrs;**
**u_int sendsize, recvsize;**
**char *handle;**
**int (*readit) (), (*writeit) ();**

This routine initializes the XDR stream object pointed to by *xdrs*. The stream's data is written to a buffer of size *sendsize*; a value of zero indicates the system should use a suitable default. The stream's data is read from a buffer of size *recvsize*; it too can be set to a suitable default by passing a zero value. When a stream's output buffer is full, *writeit* is called. Similarly, when a stream's input buffer is empty, *readit* is called. The behavior of these two routines is similar to the system calls **read** and **write**, except that *handle* is passed to the former routines as the first parameter. Note: the XDR stream's *op* field must be set by the caller.

Warning: this XDR stream implements an intermediate record stream. Therefore there are additional bytes in the stream to provide record boundary information.

**xdrrec_endofrecord(xdrs, sendnow)**
**XDR *xdrs;**
**int sendnow;**

This routine can be invoked only on streams created by **xdrrec_create()**. The data in the output buffer is marked as a completed record, and the output buffer is optionally written out if *sendnow* is non-zero. This routine returns one if it succeeds, zero otherwise.

**xdrrec_eof(xdrs)**
**XDR *xdrs;**
**int empty;**

This routine can be invoked only on streams created by **xdrrec_create()**. After consuming the rest of the current record in the stream, this routine returns one if the stream has no more input, zero otherwise.

**xdrrec_skiprecord(xdrs)**
**XDR *xdrs;**

This routine can be invoked only on streams created by **xdrrec_create()**. It tells the XDR implementation that the rest of the current record in the stream's input buffer should be discarded. This routine returns one if it succeeds, zero otherwise.

**xdr_reference(xdrs, pp, size, proc)**
**XDR *xdrs;**
**char **pp;**
**u_int size;**
**xdrproc_t proc;**

A primitive that provides pointer chasing within structures. The parameter *pp* is the address of the pointer; *size* is the *sizeof* the structure that **pp* points to; and *proc* is an XDR procedure that filters the structure between its C form and its external representation. This routine returns one if it succeeds, zero otherwise.

Warning: this routine does not understand NULL pointers. Use **xdr_pointer()** instead.

**xdr_setpos(xdrs, pos)**
**XDR *xdrs;**
**u_int pos;**

A macro that invokes the set position routine associated with the XDR stream *xdrs*. The parameter *pos* is a position value obtained from **xdr_getpos()**. This routine returns one if the XDR stream could be repositioned, and zero otherwise.

Warning: it is difficult to reposition some types of XDR streams, so this routine may fail with one type of stream and succeed with another.

**xdr_short(xdrs, sp)**
**XDR *xdrs;**
**short *sp;**

A filter primitive that translates between C **short** integers and their external representations. This routine returns one if it succeeds, zero otherwise.

**void**
**xdrstdio_create(xdrs, file, op)**
**XDR *xdrs;**
**FILE *file;**
**enum xdr_op op;**

This routine initializes the XDR stream object pointed to by *xdrs*. The XDR stream data is written to, or read from, the Standard **I/O** stream *file*. The parameter *op* determines the direction of the XDR stream (either **XDR_ENCODE**, **XDR_DECODE**, or **XDR_FREE**).

Warning: the destroy routine associated with such XDR streams calls **fflush()** on the *file* stream, but never **fclose()**.

**xdr_string(xdrs, sp, maxsize)**
**XDR**
***xdrs;**
**char **sp;**
**u_int maxsize;**

A filter primitive that translates between C strings and their corresponding external representations. Strings cannot be longer than *maxsize*. Note: *sp* is the address of the string's pointer. This routine returns one if it succeeds, zero otherwise.

**xdr_u_char(xdrs, ucp)**
**XDR *xdrs;**
**unsigned char *ucp;**

A filter primitive that translates between **unsigned** C characters and their external representations. This routine returns one if it succeeds, zero otherwise.

**xdr_u_int(xdrs, up)**
**XDR *xdrs;**
**unsigned *up;**

A filter primitive that translates between C **unsigned** integers and their external representations. This routine returns one if it succeeds, zero otherwise.

**xdr_u_long(xdrs, ulp)**
**XDR *xdrs;**
**unsigned long *ulp;**

A filter primitive that translates between C **unsigned long** integers and their external representations. This routine returns one if it succeeds, zero otherwise.

**xdr_u_short(xdrs, usp)**
**XDR *xdrs;**
**unsigned short *usp;**

A filter primitive that translates between C **unsigned short** integers and their external representations. This routine returns one if it succeeds, zero otherwise.

**xdr_union(xdrs, dscmp, unp, choices, dfault)**
**XDR *xdrs;**
**int *dscmp;**
**char *unp;**
**struct xdr_discrim *choices;**
**bool_t (*defaultarm) (); /* may equal NULL */**

A filter primitive that translates between a discriminated C **union** and its corresponding external representation. It first translates the discriminant of the union located at *dscmp*. This discriminant is always an **enum_t**. Next the union located at *unp* is translated. The parameter *choices* is a pointer to an array of **xdr_discrim()** structures. Each structure contains an ordered pair of [*value*,*proc*]. If the union's discriminant is equal to the associated *value*, then the *proc* is called to translate the union. The end of the **xdr_discrim()** structure array is denoted by a routine of value NULL. If the discriminant is not found in the *choices* array, then the *defaultarm* procedure is called (if it is not NULL). Returns one if it succeeds, zero otherwise.

**xdr_vector(xdrs, arrp, size, elsize, elproc)**
**XDR *xdrs;**
**char *arrp;**
**u_int size, elsize;**
**xdrproc_t elproc;**

A filter primitive that translates between fixed-length arrays and their corresponding external representations. The parameter *arrp* is the address of the pointer to the array, while *size* is is the element count of the array. The parameter *elsize* is the *sizeof* each of the array's elements, and *elproc* is an XDR filter that translates between the array elements' C form, and their external representation. This routine returns one if it succeeds, zero otherwise.

**xdr_void()**

This routine always returns one. It may be passed to RPC routines that require a function parameter, where nothing is to be done.

**xdr_wrapstring(xdrs, sp)**
**XDR *xdrs;**
**char **sp;**

A primitive that calls **xdr_string(xdrs, sp,MAXUN.UNSIGNED );** where **MAXUN.UNSIGNED** is the maximum value of an unsigned integer. **xdr_wrapstring()** is handy because the RPC package passes a maximum of two XDR routines as parameters, and **xdr_string()**, one of the most frequently used primitives, requires three. Returns one if it succeeds, zero otherwise.

SEE ALSO

**rpc**(3N)

*Network Programming*

NAME

ypclnt, yp_get_default_domain, yp_bind, yp_unbind, yp_match, yp_first, yp_next, yp_all, yp_order, yp_master, yperr_string, ypprot_err – Yellow Pages client interface

SYNOPSIS AND DESCRIPTION

This package of functions provides an interface to the Yellow Pages (YP) network lookup service. The package can be loaded from the standard library, **/usr/lib/libc.a**. Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the Yellow Pages, including the definitions of **map** and **domain**, and a description of the various servers, databases, and commands that comprise the YP.

All input parameters names begin with *in*. Output parameters begin with *out*. Output parameters of type **char **** should be addresses of uninitialized character pointers. Memory is allocated by the YP client package using **malloc**(3), and may be freed if the user code has no continuing need for it. For each *outkey* and *outval*, two extra bytes of memory are allocated at the end that contain NEWLINE and NULL, respectively, but these two bytes are not reflected in *outkeylen* or *outvallen*. *indomain* and *inmap* strings must be non-NULL and NULL-terminated. String parameters which are accompanied by a count parameter may not be NULL, but may point to NULL strings, with the count parameter indicating this. Counted strings need not be NULL-terminated.

All functions in this package of type *int* return 0 if they succeed, and a failure code (**YPERR_***xxxx*) otherwise. Failure codes are described under DIAGNOSTICS below.

**yp_bind (indomain);**
**char *indomain;**

To use the YP services, the client process must be "bound" to a YP server that serves the appropriate domain using **yp_bind()**. Binding need not be done explicitly by user code; this is done automatically whenever a YP lookup function is called. **yp_bind()** can be called directly for processes that make use of a backup strategy (for example, a local file) in cases when YP services are not available.

**void**
**yp_unbind (indomain)**
**char *indomain;**

Each binding allocates (uses up) one client process socket descriptor; each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. **yp_unbind()** is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to **yp_unbind()** make the domain *unbound*, and free all per-process and per-node resources used to bind it.

If an RPC failure results upon use of a binding, that domain will be unbound automatically. At that point, the **ypclnt** layer will retry forever or until the operation succeeds, provided that **ypbind** is running, and either

a) the client process cannot bind a server for the proper domain, or

b) RPC requests to the server fail.

If an error is not RPC-related, or if **ypbind** is not running, or if a bound **ypserv** process returns any answer (success or failure), the ypclnt layer will return control to the user code, either with an error code, or a success code and any results.

**yp_get_default_domain (outdomain);**
**char **outdomain;**

The YP lookup calls require a map name and a domain name, at minimum. It is assumed that the client process knows the name of the map of interest. Client processes should fetch the node's default domain by calling **yp_get_default_domain( )**, and use the returned *outdomain* as the *indomain* parameter to successive YP calls.

**yp_match(indomain, inmap, inkey, inkeylen, outval, outvallen)**
**char *indomain;**
**char *inmap;**
**char *inkey;**
**int inkeylen;**
**char **outval;**
**int *outvallen;**

**yp_match( )** returns the value associated with a passed key. This key must be exact; no pattern matching is available.

**yp_first(indomain, inmap, outkey, outkeylen, outval, outvallen)**
**char *indomain;**
**char *inmap;**
**char **outkey;**
**int *outkeylen;**
**char **outval;**
**int *outvallen;**

**yp_first( )** returns the first key-value pair from the named map in the named domain.

**yp_next(indomain, inmap, inkey, inkeylen, outkey, outkeylen, outval, outvallen);**
**char *indomain;**
**char *inmap;**
**char *inkey;**
**int inkeylen;**
**char **outkey;**
**int *outkeylen;**
**char **outval;**
**int *outvallen;**

**yp_next( )** returns the next key-value pair in a named map. The *inkey* parameter should be the *outkey* returned from an initial call to **yp_first( )** (to get the second key-value pair) or the one returned from the nth call to **yp_next( )** (to get the nth + second key-value pair).

The concept of first (and, for that matter, of next) is particular to the structure of the YP map being processing; there is no relation in retrieval order to either the lexical order within any original (non-YP) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs. The only ordering guarantee made is that if the **yp_first( )** function is called on a particular map, and then the **yp_next( )** function is repeatedly called on the same map at the same server until the call fails with a reason of YPERR_NOMORE, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

```
yp_all(indomain, inmap, incallback);
char *indomain;
char *inmap;
struct ypall_callback incallback;
```

**yp_all()** provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction take place as a single RPC request and response. You can use **yp_all()** just like any other YP procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. You return from the call to **yp_all()** only when the transaction is completed (successfully or unsuccessfully), or your foreach function decides that it does not want to see any more key-value pairs.

The third parameter to **yp_all()** is

```
struct ypall_callback *incallback {
int (*foreach)();
char *data;
};
```

The function **foreach** is called

```
foreach(instatus, inkey, inkeylen, inval, invallen, indata);
int instatus;
char *inkey;
int inkeylen;
char *inval;
int invalllen;
char *indata;
```

The *instatus* parameter will hold one of the return status values defined in **<rpcsvc/yp_prot.h>** — either **YP_TRUE** or an error code. (See **ypprot_err()**, below, for a function which converts a YP protocol error code to a ypclnt layer error code.)

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the **yp_all()** function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the **foreach** function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the **foreach** function look exactly as they do in the server's map — if they were not NEWLINE-terminated or NULL-terminated in the map, they will not be here either.

The *indata* parameter is the contents of the **incallback->data** element passed to **yp_all()**. The **data** element of the callback structure may be used to share state information between the **foreach** function and the mainline code. Its use is optional, and no part of the YP client package inspects its contents — cast it to something useful, or ignore it as you see fit.

The **foreach** function is a Boolean. It should return zero to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If **foreach** returns a non-zero value, it is not called again; the functional value of **yp_all()** is then 0.

```
yp_order(indomain, inmap, outorder);
char *indomain;
char *inmap;
int *outorder;
```

yp_order() returns the order number for a map.

```
yp_master(indomain, inmap, outname);
char *indomain;
char *inmap;
char **outname;
```

yp_master() returns the machine name of the master YP server for a map.

```
char *yperr_string(incode)
int incode;
```

yperr_string() returns a pointer to an error message string that is NULL-terminated but contains no period or NEWLINE.

```
ypprot_err (incode)
unsigned int incode;
```

ypprot_err() takes a YP protocol error code as input, and returns a ypclnt layer error code, which may be used in turn as an input to yperr_string().

FILES

/usr/include/rpcsvc/ypclnt.h
/usr/include/rpcsvc/yp_prot.h
/usr/lib/libc.a

SEE ALSO

malloc(3), ypupdate(3N), ypfiles(5), ypserv(8)

DIAGNOSTICS

All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails.

```
#define YPERR_BADARGS
        1       /* args to function are bad
#define YPERR_RPC
        2       /* RPC failure - domain has
#define YPERR_DOMAIN
        3       /* can't bind to server on
#define YPERR_MAP
        4       /* no such map in server's
#define YPERR_KEY
        5       /* no such key in map
#define YPERR_YPERR
        6       /* internal yp server or client
#define YPERR_RESRC
        7       /* resource allocation failure */
#define YPERR_NOMORE
        8       /* no more records in map
#define YPERR_PMAP
        9       /* can't communicate with portmapper */
#define YPERR_YPBIND
```

```
        10      /* can't communicate with ypbind */
#define YPERR_YPSERV
        11      /* can't communicate with ypserv */
#define YPERR_NODOM
        12      /* local domain name not set
#define  YPERR_BADDBfR
        13      /* yp database is bad */
#define  YPERR_VERSfR
        14      /* yp version mismatch */
#define  YPERR_ACCESS
        15      /* access violation */
#define  YPERR_BUSY
        16      /* database busy */
```

NAME

yp_update – changes yp information

SYNOPSIS

```
#include <rpcsvc/ypclnt.h>

yp_update(domain, map, ypop, key, keylen, data, datalen)
        char *domain;
        char *map;
        unsigned ypop
        char *key;
        int keylen;
        char *data;
        int datalen;
```

DESCRIPTION

**yp_update()** is used to make changes to the YP database. The syntax is the same as that of **yp_match()** except for the extra parameter *ypop* which may take on one of four values. If it is **YPOP_CHANGE** then the data associated with the key will be changed to the new value. If the key is not found in the database, then **yp_update()** will return **YPERR_KEY**. If *ypop* has the value **YPOP_INSERT** then the key-value pair will be inserted into the database. The error **YPERR_KEY** is returned if the key already exists in the database. To store an item into the database without concern for whether it exists already or not, pass *ypop* as **YPOP_STORE** and no error will be returned if the key already or does not exist. To delete an entry, the value of *ypop* should be **YPOP_DELETE**.

This routine depends upon secure RPC, and will not work unless the network is running secure RPC.

SEE ALSO

*System and Network Administration*

## NAME

intro – introduction to the lightweight process library (LWP)

## DESCRIPTION

The lightweight process library (LWP) provides a mechanism to support multiple threads of control that share a single address space. Under SunOS, the address space is derived from a single *forked* ("heavy-weight") process. Each thread has its own stack segment (specified when the thread is created) so that it can access local variables and make procedure calls independently of other threads. The collection of threads sharing an address space is called a *pod*. Under SunOS, threads share all of the resources of the heavyweight process that contains the pod, including descriptors and signal handlers.

The LWP provides a means for creating and destroying threads, message exchange between threads, manipulating condition variables and monitors, handling synchronous exceptions, mapping asynchronous events into messages, mapping synchronous events into exceptions, arranging for special per-thread context, multiplexing the clock for timeouts, and scheduling threads both preemptively and non–preemptively.

The LWP system exists as a library of routines (**/usr/lib/liblwp.a**) linked in (**–llwp**) with a client program which should **#include** the file **<lwp/lwp.h>**. **main** is transparently converted into a lightweight process as soon as it attempts to use any LWP primitives.

When an object created by a LWP primitive is destroyed, every attempt is made to clean up after it. For example, if a thread dies, all threads blocked on sends to or receives from that thread are unblocked, and all monitor locks held by the dead thread are released.

Because there is no kernel support for threads at present, system calls effectively block the entire pod. By linking in the non-blocking I/O library (**–lnbio**) ahead of the LWP library, you can alleviate this problem for those system calls that can issue a signal when a system call would be profitable to try. This library (which redefines some system calls) uses asynchronous I/O and events (for example, **SIGCHLD** and **SIGIO**) to make blocking less painful. The system calls remapped by the nbio library are: **open**(2V), **socket**(2), **pipe**(2), **close**(2), **read**(2V), **write**(2V), **send**(2), **recv**(2), **accept**(2), **connect**(2), **select**(2), **wait**(2)

## RETURN VALUES

LWP primitives return –1 on errors. Upon success, a non-negative integer is returned. See **lwp_perror**(3L) for details on error handling.

## FILES

**/usr/lib/liblwp.a**
**/usr/lib/libnbio.a**
**/usr/include/lwp/check.h**
**/usr/include/lwp/lwp.h**
**/usr/include/lwp/lwperror.h**
**/usr/include/lwp/lwpmachdep.h**
**/usr/include/lwp/stackdep.h**

## SEE ALSO

**accept**(2), **close**(2), **connect**(2), **open**(2V), **pipe**(2), **read**(2V), **recv**(2), **select**(2), **send**(2), **socket**(2), **wait**(2) **write**(2V),
*Lightweight Processes* in the *System Services Overview*

## INDEX

The following are the primitives currently supported, grouped roughly by function.

### Thread Creation

**lwp_self(tid)**
**lwp_getstate(tid, statvec)**
**lwp_setregs(tid, machstate)**
**lwp_getregs(tid, machstate)**
**lwp_ping(tid)**
**lwp_create(tid, pc, prio, flags, stack, nargs, arg1, ..., argn)**
**lwp_destroy(tid)**

**lwp_enumerate(vec, maxsize)**
**pod_setexit(status)**
**pod_getexit()**
**pod_exit(status)**
**SAMETHREAD(t1, t2)**

Thread Scheduling

**pod_setmaxpri(maxprio)**
**pod_getmaxpri()**
**pod_getmaxsize()**
**lwp_resched(prio)**
**lwp_setpri(tid, prio)**
**lwp_sleep(timeout)**
**lwp_suspend(tid)**
**lwp_resume(tid)**
**lwp_yield(tid)**
**lwp_join(tid)**

Error Handling

**lwp_geterr()**
**lwp_perror(s)**
**lwp_errstr()**

Messages

**msg_send(tid, argbuf, argsize, resbuf, ressize)**
**msg_recv(tid, argbuf, argsize, resbuf, ressize, timeout)**
**MSG_RECVALL(tid, argbuf, argsize, resbuf, ressize, timeout)**
**msg_reply(tid)**
**msg_enumsend(vec, maxsize)**
**msg_enumrecv(vec, maxsize)**

Event Mapping (Agents)

**agt_create(agt, event, memory)**
**agt_enumerate(vec, maxsize)**
**agt_trap(event)**

Thread Synchronization: Monitors

**mon_create(mid)**
**mon_destroy(mid)**
**mon_enter(mid)**
**mon_exit(mid)**
**mon_enumerate(vec, maxsize)**
**mon_waiters (mid, owner, vec, maxsize)**
**mon_cond_enter(mid)**
**mon_break(mid)**
**MONITOR(mid)**
**SAMEMON(m1, m2)**

Thread Synchronization: Condition Variables

**cv_create(cv, mid)**
**cv_destroy(cv)**
**cv_wait(cv)**
**cv_notify(cv)**
**cv_send(cv, tid)**
**cv_broadcast(cv)**
**cv_enumerate(vec, maxsize)**
**cv_waiters(cv, vec, maxsize)**
**SAMECV(c1, c2)**

**Exception Handling**

**exc_handle(pattern, func, arg)**
**exc_unhandle( )**
**(*exc_bound(pattern, arg))( )**
**exc_notify(pattern)**
**exc_raise(pattern)**
**exc_on_exit(func, arg)**
**exc_uniqpatt( )**

**Special Context Handling**

**lwp_ctxinit(tid, cookie)**
**lwp_ctxremove(tid, cookie)**
**lwp_ctxset(save, restore, ctxsize, optimise)**
**lwp_ctxmemget(mem, tid, ctx)**
**lwp_ctxmemset(mem, tid, ctx)**
**lwp_fpset(tid)**
**lwp_libcset(tid)**

**Stack Management**

**CHECK(location, result)**
**lwp_setstkcache(minsize, numstks)**
**lwp_newstk( )**
**lwp_datastk(data, size, addr)**
**lwp_stkcswset(tid, limit)**
**lwp_checkstkset(tid, limit)**
**STKTOP(s)**

## BUGS

There is no language support available from C.
There is no kernel support yet. Thus system calls in different threads cannot execute in parallel.
Killing a process that uses the non-blocking I/O library may leave objects (such as its standard input) in a non-blocking state. This could cause confusion to the shell.

LIST OF LWP LIBRARY FUNCTIONS

| Name | Appears on Page | Description |
|---|---|---|
| **agt_create()** | **agt_create**(3L) | map LWP events into messages |
| **agt_enumerate()** | **agt_create**(3L) | map LWP events into messages |
| **agt_trap()** | **agt_create**(3L) | map LWP events into messages |
| **CHECK()** | **lwp_newstk**(3L) | LWP stack management |
| **cv_broadcast()** | **cv_create**(3L) | manage LWP condition variables |
| **cv_create()** | **cv_create**(3L) | manage LWP condition variables |
| **cv_destroy()** | **cv_create**(3L) | manage LWP condition variables |
| **cv_enumerate()** | **cv_create**(3L) | manage LWP condition variables |
| **cv_notify()** | **cv_create**(3L) | manage LWP condition variables |
| **cv_send()** | **cv_create**(3L) | manage LWP condition variables |
| **cv_wait()** | **cv_create**(3L) | manage LWP condition variables |
| **cv_waiters()** | **cv_create**(3L) | manage LWP condition variables |
| **exc_bound()** | **exc_handle**(3L) | LWP exception handling |
| **exc_handle()** | **exc_handle**(3L) | LWP exception handling |
| **exc_notify()** | **exc_handle**(3L) | LWP exception handling |
| **exc_on_exit()** | **exc_handle**(3L) | LWP exception handling |
| **exc_raise()** | **exc_handle**(3L) | LWP exception handling |
| **exc_unhandle()** | **exc_handle**(3L) | LWP exception handling |
| **exc_uniqpatt()** | **exc_handle**(3L) | LWP exception handling |
| **lwp_checkstkset()** | **lwp_newstk**(3L) | LWP stack management |
| **lwp_create()** | **lwp_create**(3L) | LWP thread creation and destruction primitives |
| **lwp_ctxinit()** | **lwp_ctxinit**(3L) | special LWP context operations |
| **lwp_ctxremove()** | **lwp_ctxinit**(3L) | special LWP context operations |
| **lwp_ctxset()** | **lwp_ctxinit**(3L) | special LWP context operations |
| **lwp_ctxmemget()** | **lwp_ctxinit**(3L) | special LWP context operations |
| **lwp_ctxmemset()** | **lwp_ctxinit**(3L) | special LWP context operations |
| **lwp_destroy()** | **lwp_create**(3L) | LWP thread creation and destruction primitives |
| **lwp_enumerate()** | **lwp_status**(3L) | LWP status information |
| **lwp_errstr()** | **lwp_perror**(3L) | LWP error handling |
| **lwp_fpset()** | **lwp_ctxinit**(3L) | special LWP context operations |
| **lwp_geterr()** | **lwp_perror**(3L) | LWP error handling |
| **lwp_getstate()** | **lwp_status**(3L) | LWP status information |
| **lwp_setregs()** | **lwp_status**(3L) | LWP status information |
| **lwp_getregs()** | **lwp_status**(3L) | LWP status information |
| **lwp_ping()** | **lwp_status**(3L) | LWP status information |
| **lwp_join()** | **lwp_yield**(3L) | control LWP scheduling |
| **lwp_libcset()** | **lwp_ctxinit**(3L) | special LWP context operations |
| **lwp_newstk()** | **lwp_newstk**(3L) | LWP stack management |
| **lwp_datastk()** | **lwp_newstk**(3L) | LWP stack management |
| **lwp_perror()** | **lwp_perror**(3L) | LWP error handling |
| **lwp_resched()** | **lwp_yield**(3L) | control LWP scheduling |
| **lwp_resume()** | **lwp_yield**(3L) | control LWP scheduling |
| **lwp_self()** | **lwp_status**(3L) | LWP status information |
| **lwp_setpri()** | **lwp_yield**(3L) | control LWP scheduling |
| **lwp_setstkcache()** | **lwp_newstk**(3L) | LWP stack management |
| **lwp_sleep()** | **lwp_yield**(3L) | control LWP scheduling |
| **lwp_stkcswset()** | **lwp_newstk**(3L) | LWP stack management |
| **lwp_suspend()** | **lwp_yield**(3L) | control LWP scheduling |
| **lwp_yield()** | **lwp_yield**(3L) | control LWP scheduling |
| **MINSTACKSZ()** | **lwp_newstk**(3L) | LWP stack management |

| | | |
|---|---|---|
| **mon_break()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **mon_cond_enter()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **mon_create()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **mon_destroy()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **mon_enter()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **mon_enumerate()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **mon_exit()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **mon_waiters()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **MONITOR()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **msg_enumrecv()** | **msg_send**(3L) | LWP send and receive messages |
| **msg_enumsend()** | **msg_send**(3L) | LWP send and receive messages |
| **msg_recv()** | **msg_send**(3L) | LWP send and receive messages |
| **MSG_RECVALL()** | **msg_send**(3L) | LWP send and receive messages |
| **msg_reply()** | **msg_send**(3L) | LWP send and receive messages |
| **msg_send()** | **msg_send**(3L) | LWP send and receive messages |
| **pod_exit()** | **lwp_create**(3L) | LWP thread creation and destruction primitives |
| **pod_getexit()** | **lwp_create**(3L) | LWP thread creation and destruction primitives |
| **pod_getmaxpri()** | **pod_setmaxpri**(3L) | control LWP scheduling priority |
| **pod_getmaxsize()** | **pod_setmaxpri**(3L) | control LWP scheduling priority |
| **pod_setexit()** | **lwp_create**(3L) | LWP thread creation and destruction primitives |
| **pod_setmaxpri()** | **pod_setmaxpri**(3L) | control LWP scheduling priority |
| **SAMECV()** | **cv_create**(3L) | manage LWP condition variables |
| **SAMEMON()** | **mon_create**(3L) | LWP routines to manage critical sections |
| **SAMETHREAD()** | **lwp_create**(3L) | LWP thread creation and destruction primitives |
| **STKTOP()** | **lwp_newstk**(3L) | LWP stack management |

NAME

agt_create, agt_enumerate, agt_trap – map LWP events into messages

SYNOPSIS

```
#include <lwp/lwp.h>

thread_t
agt_create(agt, event, memory)
thread_t *agt;
int event;
caddr_t memory;

int
agt_enumerate(vec, maxsize)
thread_t vec[ ];
int maxsize;

int
agt_trap(event)
int event;
```

DESCRIPTION

Agents are entities that act like threads sending messages when an asynchronous event occurs. **agt_create( )** creates an object called an *agent* which maps the asynchronous event *event* into messages that can be received with **msg_receive**. *agt* stores the handle on this object. *event* is a UNIX signal number.

**agt_trap( )** causes the event, *event*, to generate an exception (see **exc_handle**(3L)). Once initialized using **agt_create( )** or **agt_trap**, an event can not be remapped to a different style of handling. If traps are enabled, an event will cause the termination of the *thread* running at the time of the trap if the trap exception is not handled. If an exception handler is in place, an exception will be raised. If an agent exists for the event, the event is mapped into a message for the agent. If neither agent nor trap mapping is enabled, the default signal action (SIG_DFL) is applied to the *pod*. Use of standard UNIX signal handling facilities will defeat the event mapping mechanism.

The message sent by the agent (in the argument buffer) will look like any other message with the sender being the agent. The receive buffer is NULL. A message is always sent by an agent to the thread which created the agent.

All messages sent by an agent contain an **eventinfo_t**. This structure indicates the thread running at the time the interrupt happened, and the particular event that occurred. Some agent messages contain more information if the particular event warrants it. In this case, a struct containing an **eventinfo_t** as its first element is passed as the argument buffer. Definitions of these structures are contained in **<lwp/lwp.h>**.

An agent appears to the owning thread just like another thread. It must therefore have some memory for holding its message, as the sender and receiver must belong to the same address space. *memory* is the space an agent will use to store its message. Typically, this is on the stack of the thread that created the agent. It must be of the correct size for the kind of event being created (most events need something to store an **eventinfo_t**. **SIGCHLD** events need room for a **sigchldev_t**.)

You should reply to an agent (using **msg_reply** (see **msg_send**(3L)) as you would reply to a thread. Although agents do not ordinarily lose events, the next agent message will not be delivered until a reply is sent to the agent. Thus, an agent appears to the client as an ordinary thread sending messages. An agent will only lose events if the total number of unreplied-to events in a pod exceeds AGENTMEMORY.

**lwp_destroy( )** is used to destroy an agent. All agents created by a thread automatically disappear when that thread dies. **agt_enumerate( )** fills in a list with the ID's of all existing agents and returns the total number of agents. This primitive uses *maxsize* to avoid exceeding the capacity of the list. If the number of agents is greater than *maxsize*, only *maxsize* agents ID's are filled in *vec*. If *maxsize* is zero, **agt_enumerate( )** returns the total number of agents.

The special event **LASTRITES** is caused by the termination of a thread. An agent for **LASTRITES** will be informed about every thread that terminates, regardless of cause. The **eventinfo_code** element of this agent will contain the stack argument that the dead thread was created with. Note: by allocating adjacent space above the thread stack, this argument can be used to point to private information about a thread. The **eventinfo_victimid** element will contain the id of the dead thread.

## RETURN VALUE

Upon successful completion, **agt_create()** and **agt_trap()** return 0. Otherwise, −1 is returned.

**agt_enumerate()** returns the total number of agents.

## ERRORS

**agt_trap()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_INVALIDARG | Event specified does not exist. |
| LE_INUSE | Agent in use for this event. |

**agt_create()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_INVALIDARG | Attempt to create agent for non-existent event. |
| LE_INUSE | Trap mapping in use for this event. |

## SEE ALSO

**exc_handle**(3L), **msg_send**(3L)

## BUGS

Signal handlers always take the **SIG_DFL** action when no agent manages the event.

If a descriptor used by a parent of the pod (such as its standard input) is marked non-blocking by a thread, it should be reset when the pod terminates to prevent the parent from receiving EWOULDBLOCK errors on the descriptor. There is no way to prevent this from happening if a pod is terminated with extreme prejudice (for instance, using **SIGKILL**).

If an agent reports that a descriptor has I/O available, there may be more than one occurrence of I/O available from that descriptor. Thus, being informed that **SIGIO** has occurred on socket *s* may mean that there are several messages waiting to be received from *s*. Clients should be careful to clean out all I/O from a descriptor before going back to sleep.

All system calls should be protected with loops testing for EINTR (and monitors if multiple threads can try to use system calls concurrently). An **lwp_sleep()** could result in a hidden clock interrupt for example.

## WARNINGS

**agt_trap()** should not be used for asynchronous events. If an unsuspecting thread which has no exception handler is running at the time of a trapped event, it will be terminated.

Clients should not normally handle signals themselves since the agent mechanism assumes it is the only entity handling signals.

## NAME

cv_create, cv_destroy, cv_wait, cv_notify, cv_broadcast, cv_send, cv_enumerate, cv_waiters, SAMECV – manage LWP condition variables

## SYNOPSIS

```
#include <lwp/lwp.h>

cv_t
cv_create(cv, mid)
cv_t *cv;
mon_t mid;

int
cv_destroy(cv)
cv_t cv;

int
cv_wait(cv)
cv_t cv;

int
cv_notify(cv)
cv_t cv;

int
cv_send(cv, tid)
cv_t cv;
lwp_t tid

int
cv_broadcast(cv)
cv_t cv;

int
cv_enumerate(vec, maxsize)
cv_t vec[ ];      /* will contain list of all conditions */
int maxsize;      /* maximum size of vec */

int
cv_waiters(cv, vec, maxsize)
cv_t cv;          /* condition variable being interrogated */
thread_t vec[ ];  /* which threads are blocked on cv */
int maxsize;      /* maximum size of vec */

SAMECV(c1, c2)
```

## DESCRIPTION

Condition variables are useful for synchronization within monitors. By waiting on a condition variable, the currently-held monitor (a condition variable must *always* be used within a monitor) is released atomically and the invoking thread is suspended. When monitors are nested, monitor locks other than the current one are retained by the thread. At some later point, a different thread may awaken the waiting thread by issuing a notification on the condition variable. When the notification occurs, the waiting thread will queue to reacquire the monitor it gave up. It is possible to have different condition variables operating within the same monitor to allow selectivity in waking up threads.

**cv_create()** creates a new condition variable (returned in *cv*) which is bound to the monitor specified by *mid*. It is illegal to access (using **cv_wait**, **cv_notify**, **cv_send()** or **cv_broadcast**) a condition variable from a monitor other than the one it is bound to. **cv_destroy()** removes a condition variable.

**cv_wait()** blocks the current thread and releases the monitor lock associated with the condition (which must also be the monitor lock most recently acquired by the thread). Other monitor locks held by the thread are not affected. The blocked thread is enqueued by its scheduling priority on the condition.

**cv_notify()** awakens at most one thread blocked on the condition variable and causes the awakened thread to queue for access to the monitor released at the time it waited on the condition. It can be dangerous to use **cv_notify()** if there is a possibility that the thread being awakened is one of several threads that are waiting on a condition variable and the awakened thread may not be the one intended. In this case, use of **cv_broadcast()** is recommended.

**cv_broadcast()** is the same as **cv_notify()** except that *all* threads blocked on the condition variable are awakened. **cv_notify()** and **cv_broadcast()** do nothing if no thread is waiting on the condition. For both **cv_notify()** and **cv_broadcast**, the currently held monitor must agree with the one bound to the condition by **cv_create**.

**cv_send()** is like **cv_notify()** except that the particular thread **tid** is awakened. If this thread is not currently blocked on the condition, **cv_send()** reports an error.

**cv_enumerate()** lists the ID of all of the condition variables. The value returned is the total number of condition variables. The vector supplied is filled in with the ID's of condition variables. **cv_waiters()** lists the ID's of the threads blocked on the condition variable *cv* and returns the number of threads blocked on *cv*. For both **cv_enumerate()** and **cv_waiters**, *maxsize* is used to avoid exceeding the capacity of the list *vec*. If the number of entries to be filled is greater than *maxsize*, only *maxsize* entries are filled in *vec*. It is legal in both of these primitives to specify a *maxsize* of 0.

SAMECV is a convenient predicate used to compare two condition variables for equality.

## RETURN VALUE

**cv_create, cv_destroy, cv_send, cv_wait, cv_notify, cv_broadcast:**

A 0 return indicates success; −1 indicates an error.

**cv_enumerate()** returns the total number of condition variables.

**cv_waiters()** returns the number of threads blocked on a condition variable.

## ERRORS

**cv_destroy()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_INUSE | Attempt to destroy condition variable being waited on by a thread. |
| LE_NONEXIST | Attempt to destroy non-existent condition variable. |

**cv_wait()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_NOTOWNED | Attempt to wait on a condition without possessing the correct monitor lock. |
| LE_NONEXIST | Attempt to wait on non-existent condition variable. |

**cv_notify()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_NOTOWNED | Attempt to notify condition variable without possessing the correct monitor. |
| LE_NONEXIST | Attempt to notify non-existent condition variable. |

**cv_send()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_NOTOWNED | Attempt to awaken condition variable without possessing the correct monitor lock. |
| LE_NONEXIST | Attempt to awaken non-existent condition variable. |
| LE_NOWAIT | The specified thread is not currently blocked on the condition. |

**cv_broadcast()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_NOTOWNED | Attempt to broadcast condition without possessing the correct monitor lock. |
| LE_NONEXIST | Attempt to broadcast non-existent condition variable. |

**SEE ALSO**

**mon_create**(3L)

NAME

exc_handle, exc_unhandle, exc_bound, exc_notify, exc_raise, exc_on_exit, exc_uniqpatt – LWP exception handling

SYNOPSIS

```
#include <lwp/lwp.h>

int
exc_handle(pattern, func, arg)
int pattern;
caddr_t (*func)();
caddr_t arg;

int
exc_raise(pattern)
int pattern;

int
exc_unhandle()

caddr_t
(*exc_bound(pattern, arg))()
int pattern;
caddr_t *arg;

int
exc_notify(pattern)
int pattern;

int
exc_on_exit(func, arg)
void (*func)();
caddr_t arg;

int
exc_uniqpatt()
```

DESCRIPTION

These primitives can be used to manage exceptional conditions in a thread. Basically, raising an exception is a more general form of non-local goto or *longjmp*, but the invocation is pattern-based. It is also possible to *notify* an exception handler whereby a function supplied by the exception handler is invoked and control is returned to the raiser of the exception. Finally, one can establish a handler which is always invoked upon procedure exit, regardless of whether the procedure exits using a *return* or an exception raised to a handler established prior to the invocation of the exiting procedure.

**exc_handle()** is used to establish an exception handler. **exc_handle()** returns 0 to indicate that a handler has been established. A return of −1 indicates an error in trying to establish the exception handler. If it returns something else, an exception has occurred and any procedure calls deeper than the one containing the handler have disappeared. All exception handlers established by a procedure are automatically discarded when the procedure terminates.

**exc_handle()** binds a *pattern* to the handler, where a pattern is an integer, and two patterns *match* if their values are equal. When an exception is raised with **exc_raise**, the most recent handler that has established a matching pattern will catch the exception. A special pattern (CATCHALL) is provided which matches any **exc_raise()** pattern. This is useful for handlers which know that there is no chance the resources allocated in a routine can be reclaimed by previous routines in the call chain.

The other two arguments to **exc_handle()** are a function and an argument to that function. **exc_bound()** retrieves these arguments from an **exc_handle()** call made by the specified thread. By using **exc_bound()** to retrieve and call a function bound by the exception handler, a procedure can raise a *notification exception* which allows control to return to the raiser of the exception after the exception is handled.

**exc_raise()** allows the caller to transfer control (do a non-local goto) to the matching **exc_handle**. This matching exception handler is destroyed after the control transfer. At this time, it behaves as if **exc_handle()** returns with the *pattern* from **exc_raise()** as the return value. Note: *func* of **exc_handle()** is not called using **exc_raise()** — it is only there for notification exceptions. Because the exception handler returns the pattern that invoked it, it is possible for a handler that matches the CATCHALL pattern to *reraise* the exact exception it caught by using **exc_raise()** on the caught pattern. It is illegal to handle or raise the pattern 0 or the pattern −1. Handlers are searched for pattern matches in the reverse execution order that they are set (i.e., the most recently established handler is searched first).

**exc_unhandle()** destroys the most recently established exception handler set by the current thread. It is an error to destroy an exit-handler set up by **exc_on_exit**. When a procedure exits, all handlers and exit handlers set in the procedure are automatically deallocated.

**exc_notify()** is a convenient way to use **exc_bound**. The function which is bound to *pattern* is retrieved. If the function is not NULL, the function is called with the associated argument and the result is returned. If the function is NULL, **exc_raise(***pattern***)** is returned.

**exc_on_exit()** specifies an exit procedure and argument to be passed to the exit procedure, which is called when the procedure which sets an exit handler using **exc_on_exit()** exits. The exit procedures (more than one may be set) will be called regardless if the setting procedure is exited using a *return* or an **exc_raise**. Because the exit procedure is called as if the handling procedure had returned, the argument passed to it should not contain addresses on the handler's stack. However, any value returned by the procedure which established the exit procedure is preserved no matter what the exit procedure returns. This primitive is used in the MONITOR macro to enforce the monitor discipline on procedures.

Some signals can be considered to be synchronous traps. They are usually the starred (*) signals in the **signal**(3) man pages. These are: **SIGSYS, SIGBUS, SIGEMT, SIGFPE, SIGILL, SIGTRAP, SIGSEGV.** If an event is marked as a trap using **agt_trap** (see **agt_create**(3L)) the event will generate exceptions instead of agent messages. This mapping is per-pod, not per-thread. A thread which handles the signal number of one of these as the pattern for **exc_handle()** will catch such a signal as an exception. The exception will be raised as an **exc_notify()** so either escape or notification style exceptions can be used, depending on what the matching **exc_handle()** provides. If the exception is not handled, the thread will terminate. Note: it can be dangerous to supply an exception handler to treat stack overflow since the client's stack is used in raising the exception.

**exc_uniqpatt()** returns an exception pattern that is not any of the pre-defined patterns (any of the synchronous exceptions or −1 or CATCHALL). Each call to **exc_uniqpatt()** results in a different pattern. If **exc_uniqpatt()** cannot guarantee uniqueness, −1 is returned instead the *first* time this happens. Subsequent calls after this error result in patterns which may be duplicates.

RETURN VALUE

**exc_uniqpatt()** returns −1 the *first* time it fails. Otherwise, it returns a unique pattern.

When **exc_handle()** is called, a return value of 0 indicates success and −1 indicates error. When **exc_handle()** returns because of a matching **exc_raise()** call, it returns the *pattern* raised by **exc_raise**.

Upon successful completion, **exc_raise()** transfers control to the matching **exc_handle()** and does not return. If there is an error, **exc_raise()** returns −1.

Upon successful completion, **exc_unhandle()** returns 0. It returns −1 if there is an error.

**exc_bound()** returns a pointer to a function or 0 if no function was bound.

Upon successful completion, **exc_notify()** returns the return value of a function or transfers control to a matching **exc_handle()** and does not return. It returns −1 if there is an error.

**exc_on_exit()** returns 0.

ERRORS

**exc_unhandle( )** will fail if one or more of the following is true:

LE_NONEXIST Attempt to remove an exit handler or a non-existent handler.

**exc_raise( )** will fail if one or more of the following is true:

LE_NONEXIST No context found to raise an exception to.

LE_INVALIDARG Attempt to raise an illegal pattern (–1 or 0).

**exc_handle( )** will fail if one or more of the following is true:

LE_INVALIDARG Attempt to handle an illegal pattern (–1 or 0).

**exc_uniqpatt( )** will fail if one or more of the following is true:

LE_REUSE Possible reuse of existing object. **agt_create**(3L)

BUGS

The stack may not contain useful information after an exception has been caught so post-exception debugging can be difficult. The reason for this is that a given handler may call procedures that trash the stack before reraising an exception.

The distinction between traps and interrupts can be problematical.

The environment restored on **exc_raise( )** consists of the registers at the time of the **exc_handle**. As a result, modifications to register variables between the times of **exc_handle( )** and **exc_raise( )** will not be seen. This problem does not occur in the sun4 implementation.

WARNINGS

**exc_on_exit( )** passes a simple type as an argument to the exit routine. If you need to pass a complex type, such as a *thread_t*, *mon_t*, or *cv_t*, pass a pointer to the object instead.

NAME

lwp_create, lwp_destroy, SAMETHREAD, pod_setexit, pod_getexit, pod_exit – LWP thread creation and destruction primitives

SYNOPSIS

```
#include <lwp/lwp.h>
#include <lwp/stackdep.h>

int
lwp_create(tid, func, prio, flags, stack, nargs, arg1, ..., argn)
thread_t *tid;
void (*func)();
int prio;
int flags;
stkalign_t *stack;
int nargs;
int arg1, ..., argn;

int
lwp_destroy(tid)
thread_t tid;

void
pod_setexit(status)
int status;

int
pod_getexit(status)
int status;

void
pod_exit(status)
int status

SAMETHREAD(t1, t2)
```

DESCRIPTION

**lwp_create()** creates a lightweight process which starts at address *func* and has stack segment *stack*. If *stack* is NULL, the thread is created in a suspended state (see below) and no stack or pc is bound to the thread. *prio* is the scheduling priority of the thread (higher priorities are favored by the scheduler). The identity of the new thread is filled in the reference parameter *tid*. *flags* describes some options on the new thread. **LWPSUSPEND** creates the thread in suspended state (see **lwp_yield**(3L)). **LWPNOLASTRITES** will disable the **LASTRITES** agent message when the thread dies. The default (0) is to create the thread in running state with **LASTRITES** reporting enabled. **LWPSERVER** indicates that a thread is only viable as long as non-LWPSERVER threads are alive. The pod will terminate if the only living threads are marked **LWPSERVER** and blocked on a lwp resource (for instance, waiting for a message to be sent). *nargs* is the number (0 or more) of simple-type (int) arguments supplied to the thread.

The first time a lwp primitive is used, the lwp library automatically converts the caller (i.e., **main**) into a thread with the highest available scheduling priority (see **pod_setmaxpri**(3L)). The identity of this thread can be retrieved using **lwp_self** (see **lwp_status**(3L)). This thread has the normal SunOS stack given to any *forked* process.

Scheduling is, by default, non-preemptive within a priority, and within a priority, threads enter the run queue on a FIFO basis (that is, whenever a thread becomes eligible to run, it goes to the end of the run queue of its particular priority). Thus, a thread continues to run until it voluntarily relinquishes control or an event (including thread creation) occurs to enable a higher priority thread. Some primitives may cause the current thread to block, in which case the unblocked thread with the highest priority runs next. When several threads are created with the same priority, they are queued for execution in the order of creation.

This order may not be preserved as threads yield and block within a priority. If an agent owned by a thread with a higher priority is invoked, that thread will preempt the currently running one.

There is no concept of ancestry in threads: the creator of a thread has no special relation to the thread it created. When all threads have died, the pod terminates.

**lwp_destroy()** is a way to explicitly terminate a thread or agent (instead of having an executing thread "fall though", which also terminates the thread). *tid* specifies the id of the thread or agent to be terminated. If *tid* is **SELF**, the invoking thread is destroyed. Upon termination, the resources (messages, monitor locks, agents) owned by the thread are released, in some cases resulting in another thread being notified of the death of its peer (by having a blocking primitive become unblocked with an error indication). A thread may terminate itself explicitly, although self-destruction is automatic when it returns from the procedure specified in the **lwp_create()** primitive.

**pod_setexit()** sets the exit status for a pod. This value will be returned to the parent process of the pod when the pod dies (default is 0). **exit**(3) terminates the current *thread*, using the argument supplied to *exit* to set the current value of the exit status. **on_exit**(3) establishes an action that will be taken when the entire pod terminates. **pod_exit()** is available to terminate the pod immediately with the final actions established by **on_exit**. If you wish to terminate the pod immediately, **pod_exit()** or **exit**(2) should be used. **pod_getexit()** returns the current value of the pod's exit status.

**SAMETHREAD** is a convenient predicate used to compare two threads for equality.

## RETURN VALUE

Upon successful completion, **lwp_create**, and **lwp_destroy()** return 0. Otherwise, −1 is returned. **pod_getexit()** returns the current exit status of the pod.

## ERRORS

**lwp_create()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_NOROOM | Unable to allocate memory for thread context. |
| LE_INVALIDARG | Too many arguments (> 512). |
| LE_ILLPRIO | Illegal priority. |

**lwp_destroy()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_NONEXIST | Attempt to destroy a thread or agent that does not exist. |

## SEE ALSO

**exit**(3), **lwp_yield**(3L), **on_exit**(3), **pod_setmaxpri**(3L)

## WARNINGS

Some special threads may be created silently by the lwp library. These include an *idle* thread that runs when no other activity is going on, and a *reaper* thread that frees stacks allocated by **lwp_newstk**. These special threads will show up in status calls. A pod will terminate if these special threads are the only ones extant.

NAME

lwp_ctxinit, lwp_ctxremove, lwp_ctxset, lwp_ctxmemget, lwp_ctxmemset, lwp_fpset, lwp_libcset – special LWP context operations

SYNOPSIS

```
#include <lwp/lwp.h>

int
lwp_ctxset(save, restore, ctxsize, optimise)
void (*save)(/* caddr_t ctx, thread_t old, thread_t new */);
void (*restore)(/* caddr_t ctx, thread_t old, thread_t new */);
unsigned int ctxsize;
int optimise;

int
lwp_ctxinit(tid, cookie)
thread_t tid;           /* thread with special contexts */
int cookie;             /* type of context */

int
lwp_ctxremove(tid, cookie)
thread_t tid;
int cookie;

int
lwp_ctxmemget(mem, tid, ctx)
caddr_t mem;
thread_t tid;
int ctx;

int
lwp_ctxmemset(mem, tid, ctx)
caddr_t mem;
thread_t tid;
int ctx;

int
lwp_fpset(tid)
thread_t tid;           /* thread utilizing floating point hardware */

int
lwp_libcset(tid)
thread_t tid;           /* thread utilizing errno */
```

DESCRIPTION

Normally on a context switch, only machine registers are saved/restored to provide each thread its own virtual machine. However, there are other hardware and software resources which can be multiplexed in this way. For example, floating point registers can be used by several threads in a pod. As another example, the global value **errno** in the standard C library may be used by all threads making system calls.

To accommodate the variety of contexts that a thread may need without requiring all threads to pay for unneeded switching overhead, **lwp_ctxinit( )** is provided. This primitive allows a client to specify that a given thread requires certain context to be saved and restored across context switches (by default just the machine registers are switched). More than one special context may be given to a thread.

To use **lwp_ctxinit**, it is first necessary to define a special context. **lwp_ctxset( )** specifies save and restore routines, as well as the size of the context that will be used to hold the switchable state. The *save* routine will automatically be invoked when an active thread is blocked and the *restore* routine will be invoked when a blocked thread is restarted. These routines will be passed a pointer to a buffer (initialized to all 0's) of size *ctxsize* which is allocated by the LWP library and used to hold the volatile state. In addition, the

identity of the thread whose special context is being saved (*old*) and the identity of the thread being restarted (*new*) are passed in to the *save* and *restore* routines. **lwp_ctxset()** returns a cookie used by subsequent **lwp_ctxinit()** calls to refer to the kind of context just defined. If the *optimise* flag is TRUE, a special context switch action will not be invoked unless the thread resuming execution differs from the last thread to use the special context and also uses the special context. If the *optimise* flag is FALSE, the *save* routine will always be invoked immediately when the thread using this context is scheduled out and the *restore* routine will be invoked immediately when a new thread using this context is scheduled in. Note that an unoptimised special context is protected from threads which do not use the special context but which do affect the context state. **lwp_ctxremove()** can be used to remove a special context installed by **lwp_ctxinit**.

Because context switching is done by the scheduler on behalf of a thread, it is an error to use an LWP primitive in an action done at context switch time. Also, the stack used by the save and restore routines belongs to the scheduler, so care should be taken not to use lots of stack space. As a result of these restrictions, only knowledgeable users should write their own special context switching routines.

**lwp_ctxmemget** and **lwp_ctxmemset** are used to retrieve and set (respectively) the memory associated with a given special context (*ctx*) and a given thread (*tid*). *mem* is the address of client memory that will hold the context information being retrieved or set. Note that the special context *save* and *restore* routines may be NULL, so pure data may be associated with a given thread using these primitives.

Several kinds of special contexts are predefined. To allow a thread to share floating point hardware with other threads, the **lwp_fpset()** primitive is available. The floating-point hardware bound at compile-time is selected automatically. To multiplex the global variable **errno**, **lwp_libcset()** is used to have **errno** become part of the context of thread *tid*.

Special contexts can be used to assist in managing stacks. See **lwp_newstk**(3L) for details.

## RETURN VALUE

**lwp_ctxset()** returns a cookie to be used by subsequent **lwp_ctxinit()** calls, -1 if unable to define the context.

## ERRORS

**lwp_ctxinit()** will fail if one or more of the following are true:

LE_INUSE This special context already set for this thread.

**lwp_ctxremove()** will fail if one or more of the following are true:

LE_NONEXIST The specified context is not set for this thread.

**lwp_ctxset()** will fail if one or more of the following are true:

LE_NOROOM Unable to allocate memory to define special context.

## SEE ALSO

**lwp_newstk**(3L)

## BUGS

The floating point contexts should be initialized implicitly for those threads that use floating point.

NAME

lwp_checkstkset, lwp_stkcswset, CHECK, lwp_setstkcache, lwp_newstk, lwp_datastk, STKTOP – LWP stack management

SYNOPSIS

```
#include <lwp/lwp.h>
#include <lwp/check.h>
#include <lwp/lwpmachdep.h>
#include <lwp/stackdep.h>

CHECK(location, result)

int
lwp_checkstkset(tid, limit)
thread_t tid;
caddr_t limit;

int
lwp_stkcswset(tid, limit)
thread_t tid;
caddr_t limit;

int
lwp_setstkcache(minstksz, numstks)
int minstksz;
int numstks;

stkalign_t *
lwp_newstk()

stkalign_t *
lwp_datastk(data, size, addr)
caddr_t data;
int size;
caddr_t *addr;

STKTOP(s)
```

DESCRIPTION

Stacks are problematical with lightweight processes. What is desired is that stacks for each thread are redzone protected so that one thread's stack does not unexpectedly grow into the stack of another. In addition, stacks should be of infinite length, grown as needed. The process stack is a maximum-sized segment (see **getrlimit**(2).) This stack is redzone protected, and you can even try to extend it beyond its initial maximum size in some cases. With SunOS 4.*x*, it is possible to efficiently allocate large stacks that have red zone protection, and the LWP library provides some support for this. For those systems that do not have flexible memory management, the LWP library provides assistance in dealing with the problems of maintaining multiple stacks.

The stack used by *main* is the same stack that the system allocates for a *forked* process. For allocating other thread stacks, the client is free to use any statically or dynamically allocated memory (using memory from *main*'s stack is subject to the stack resource limit for any *forked* process). In addition, the LASTRITES agent message is available to free allocated resources when a thread dies. Any stack should be at least MINSTACKSZ *stkalign_t*'s large because the LWP library will use the client stack to execute primitives. For very fast dynamically allocated stacks, a stack cacheing mechanism is available. **lwp_setstkcache**() allocates a cache of stacks. Each time the cache is empty, it is filled with *numstks* new stacks, each containing at least *minstksz* bytes. *minstksz* will automatically be augmented to take into account the stack needs of the LWP library. **lwp_newstk**() returns a cached stack that is suitable for use in an **lwp_create**() call. **lwp_setstkcache**() must be called (once) prior to any use of **lwp_newstk**. If running under SunOS 4.*x*, the stacks allocated by **lwp_newstk**() will be red-zone protected (an attempt to reference below the stack bottom will result in a SIGSEGV event).

Threads created with stacks from **lwp_newstk()** should not use the **NOLASTRITES** flag. If they do, cached stacks will not be returned to the cache when a thread dies.

**lwp_datastk()** also returns a red-zone protected stack like **lwp_newstk()** does. It copies any amount of data (subject to the size limitations imposed by **lwp_setstkcache**) onto the stack *above* the stack top that it returns. *data* points to information of *size* bytes to be copied. The exact location where the data is stored is returned in the reference parameter *addr*. Because **lwp_create()** only passes simple types to the newly-created thread, **lwp_datastk()** is useful to pass a more complex argument: Call **lwp_datastk()** to get an initialized stack, and pass the address of the data structure (*addr*) as an argument to the new thread.

A *reaper* thread running at the maximum pod priority is created by **lwp_setstkcache**. It's action may be delayed by other threads running at that priority, so it is suggested that the maximum pod priority not be used for client-created threads when **lwp_newstk()** is being used. Altering the maximum pod priority with **pod_setmaxpri()** will have the side effect of increasing the reaper thread priority as well.

The stack address passed to **lwp_create()** represents the top of the stack: the LWP library will not use any addresses at or above it. Thus, it is safe to store information above the stack top if there is room there.

For stacks that are not protected with hardware redzones, some protection is still possible. For any thread *tid* with stack boundary *limit* made part of a special context with **lwp_checkstkset**, the **CHECK** macro may be used. This macro, if used at the beginning of each procedure (and before local storage is initialized (it is okay to *declare* locals though)), will check that the stack limit has not been violated. If it has, the non-local *location* will be set to *result* and the procedure will return. **CHECK** is not perfect, as it is possible to call a procedure with many arguments after **CHECK** validates the stack, only to have these arguments clobber the stack before the new procedure is entered.

**lwp_stkcswset()** checks at context-switch time the stack belonging to thread *tid* for passing stack boundary *limit*. In addition, a checksum at the bottom of the stack is validated to ensure that the stack did not temporarily grow beyond its limit. This is automated and more efficient than using **CHECK**, but by the time a context switch occurs, it's too late to do much but **abort**(3) if the stack was clobbered.

To portably use statically allocated stacks, the macros in *stackdep.h* should be used. Declare a stack *s* to be an array of *stkalign_t*'s, and pass the stack to **lwp_create()** as **STKTOP**(s).

## RETURN VALUES

**lwp_newstk and lwp_datastk()** return 0 on failure, else a valid new stack address.

**lwp_setstkcache()** returns the actual size of the stacks allocated in the cache.

## SEE ALSO

**getrlimit**(2), **abort**(3)

## BUGS

C should provide support for heap-allocated stacks at procedure entry time. The hardware should be segment-based to eliminate the problem altogether.

## WARNING

**lwp_datastk()** should not be directly used in a **lwp_create()** call since C does not guarantee the order in which arguments to a function are evaluated.

NAME

lwp_geterr, lwp_perror, lwp_errstr – LWP error handling

SYNOPSIS

```
#include <lwp/lwp.h>
#include <lwp/lwperror.h>

lwp_err_t lwp_geterr( );

void
lwp_perror(s)
char *s;

char **lwp_errstr( );
```

DESCRIPTION

When a primitive fails (returns −1), **lwp_geterr( )** can be used to obtain the identity of the error (which is part of the context for each lwp). **lwp_perror( )** can be used to print an error message on the standard error file (analogous to **perror**(3)) when a lwp primitive returns an error indication. **lwp_perror( )** uses the same mechanism as **lwp_geterr( )** to obtain the last error. **lwp_errstr** returns a pointer to the (NULL-terminated) list of error messages.

**lwp_libcset** (see **lwp_ctxinit**(3L)) allows **errno** from the standard C library reflect a per-thread value rather than a per-pod value.

SEE ALSO

**lwp_ctxinit**(3L), **perror**(3)

NAME

lwp_self, lwp_ping, lwp_enumerate, lwp_getstate, lwp_setregs, lwp_getregs – LWP status information

SYNOPSIS

```
#include <lwp/lwp.h>
#include <lwp/lwpmachdep.h>

int
lwp_enumerate(vec, maxsize)
thread_t vec[ ];  /* list of id's to be filled in */
int maxsize;      /* number of elements in vec */

int
lwp_ping(tid)
thread_t tid;

int
lwp_getregs(tid, machstate)
thread_t tid;
machstate_t *machstate;

int
lwp_setregs(tid, machstate)
thread_t tid;
machstate_t *machstate;

int
lwp_getstate(tid, statvec)
thread_t tid;
statvec_t *statvec;

int
lwp_self(tid)
thread_t *tid;
```

DESCRIPTION

**lwp_self( )** returns the ID of the current thread in *tid*. This is the *only* way to retrieve the identity of *main*.

**lwp_enumerate( )** fills in a list with the ID's of all existing threads and returns the total number of threads. This primitive will use *maxsize* to avoid exceeding the capacity of the list. If the number of threads is greater than *maxsize*, only *maxsize* thread ID's are filled in *vec*. If *maxsize* is zero, **lwp_enumerate( )** just returns the total number of threads.

**lwp_getstate( )** is used to retrieve the context of a given thread. It is possible to see what object (thread, monitor, etc.) if any that thread is blocked on, and the scheduling priority of the thread.

**lwp_ping** returns 0 (no error) if the thread *tid* exists. Otherwise, -1 is returned.

**lwp_setregs** sets the machine-dependent context (i.e., registers) of a thread. The next time the thread is scheduled in, this context is installed. Consult **lwpmachdep.h** for the details. **lwp_getregs** retrieves the machine-dependent context. Note: the registers may not be meaningful unless the thread in question is blocked or suspended because the state of the registers as of the most recent context switch is returned.

RETURNS

Upon successful completion, **lwp_self** and **lwp_getstate( )** return 0, –1 on error.

**lwp_enumerate( )** returns the total number of threads.

**lwp_ping** returns 0 if the specified thread exists, else -1.

ERRORS

**lwp_getstate , lwp_ping ,** and **lwp_setstate( )** will fail if one or more of the following is true:

LE_NONEXIST Attempt to get the status of a non-existent thread.

NAME

lwp_yield, lwp_suspend, lwp_resume, lwp_join, lwp_setpri, lwp_resched, lwp_sleep – control LWP scheduling

SYNOPSIS

```
#include <lwp/lwp.h>

int
lwp_yield(tid)
thread_t tid;

int
lwp_sleep(timeout)
struct timeval *timeout;

int
lwp_resched(prio)
int prio;

int
lwp_setpri(tid, prio)
thread_t tid;
int prio;

int
lwp_suspend(tid)
thread_t tid;

int
lwp_resume(tid)
thread_t tid;

int
lwp_join(tid)
thread_t tid;
```

DESCRIPTION

**lwp_yield()** allows the currently running thread to voluntarily relinquish control to another thread *with the same scheduling priority.* **On a uniprocessor, it is never the case that a thread can execute while a higher priority thread is eligible to run.** If *tid* is THREADNULL, the next thread in the same priority queue of the yielding thread will run and the current thread will go the the end of the scheduling queue. Otherwise, it is the ID of the thread to run next, and the current thread will take second place in the scheduling queue.

**lwp_sleep()** blocks the thread executing this primitive for at least the time specified by *timeout*.

Scheduling of threads is, by default, preemptive (higher priorities preempt lower ones) across priorities and non-preemptive within a priority. **lwp_resched()** moves the front thread for a given priority to the end of the scheduling queue. Thus, to achieve a preemptive round–robin scheduling discipline, a high priority thread can periodically wake up and shuffle the queue of threads at a lower priority. **lwp_resched()** does not affect threads which are blocked. If the priority of the rescheduled thread is the same as that of the caller, the effect is the same as **lwp_yield**.

**lwp_setpri()** is used to alter (raise or lower) the scheduling priority of the specified thread. If *tid* is SELF, the priority of the invoking thread is set. Note: if the priority of the affected thread becomes greater than that of the caller and the affected thread is not blocked, the caller will not run next. **lwp_setpri()** can be used on either blocked or unblocked threads.

**lwp_join()** blocks the thread issuing the join until the thread *tid* terminates. More than one thread may join *tid*.

**lwp_suspend()** makes the specified thread ineligible to run. If *tid* is **SELF**, the caller is itself suspended. **lwp_resume()** undoes the effect of **lwp_suspend**. If a blocked thread is suspended, it will not run until it has been unblocked as well as explicitly made eligible to run using **lwp_resume**. By suspending a thread, one can safely examine it without worrying that its execution–time state will change.

NOTE

When scheduling preemptively, be sure to use monitors to protect shared data structures such as those used by the standard I/O library.

RETURN VALUE

*lwp_yield, lwp_sleep, lwp_resched, lwp_join, lwp_suspend, lwp_resume*:

A 0 return indicates success; −1 indicates an error.

*lwp_setpri*:

Upon successful completion, the previous priority is returned. Otherwise, −1 is returned.

ERRORS

**lwp_yield()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_INVALIDARG | Attempt to yield to a blocked thread. |
| LE_NONEXIST | Attempt to yield to a non-existent thread. |
| LE_ILLPRIO | Attempt to yield to thread with different priority. |

**lwp_sleep()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_INVALIDARG | Illegal timeout specified. |

**lwp_resched()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_INVALIDARG | Attempt to reschedule thread at priority greater than that of the caller. |
| LE_ILLPRIO | The priority queue specified contains no threads to reschedule. |

**lwp_setpri()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_INVALIDARG | The priority specified is beyond the maximum available to the pod. |
| LE_NONEXIST | Attempt to set priority of a non-existent thread. |

**lwp_join()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_NONEXIST | Attempt to join a thread that does not exist. |

**lwp_suspend()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_NONEXIST | Attempt to suspend a non-existent thread. |

**lwp_resume()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_NONEXIST | Attempt to resume a non-existent thread. |

NAME

mon_create, mon_destroy, mon_enter, mon_exit, mon_enumerate, mon_waiters, mon_cond_enter, mon_break, MONITOR, SAMEMON – LWP routines to manage critical sections

SYNOPSIS

```
#include <lwp/lwp.h>

int
mon_create(mid)
mon_t *mid;

int
mon_destroy(mid)
mon_t mid;

int
mon_enter(mid)
mon_t mid;

int
mon_exit(mid)
mon_t mid;

int
mon_enumerate(vec, maxsize)
mon_t vec[ ];     /* list of all monitors */
int maxsize;      /* max size of vec */

int
mon_waiters(mid, owner, vec, maxsize)
mon_t mid;              /* monitor in question */
thread_t *owner;        /* which thread owns the monitor */
thread_t vec[ ];        /* list of blocked threads */
int maxsize;            /* max size of vec */

int
mon_cond_enter(mid)
mon_t mid;

int
mon_break(mid)
mon_t mid;

MONITOR(mid)

SAMEMON(m1, m2)
```

DESCRIPTION

Monitors are used to synchronize access to common resources. Although it is possible (on a uniprocessor) to use knowledge of how scheduling priorities work to serialize access to a resource, monitors (and condition variables) provide a general tool to provide the necessary synchronization.

**mon_create( )** creates a new monitor and returns its identity in *mid*. **mon_destroy( )** destroys a monitor, as well as any conditions bound to it (see **cv_create**(3L)). Because the lifetime of a monitor can transcend the lifetime of the lwp that created it, monitor destruction is not automatic upon lwp destruction.

**mon_enter( )** blocks the calling thread (if the monitor is in use) until the monitor becomes free by being exited or by waiting on a condition (see **cv_create**(3L)). Threads unable to gain entry into the monitor are queued for monitor service by the priority of the thread requesting monitor access, FCFS within a priority. Monitor calls may nest. If, while holding monitor M1 a request for monitor M2 is made, M1 will be held until M2 can be acquired.

**mon_cond_enter()** will enter the monitor only if the monitor is not busy. Otherwise, an error is returned.

**mon_enter()** and **mon_cond_enter()** will allow a thread which already has the monitor to reenter the monitor. In this case, the nesting level of monitor entries is returned. Thus, the first time a monitor is entered, **mon_enter()** returns 0. The next time the monitor is entered, **mon_enter()** returns 1. **mon_exit()** frees the current monitor and allows the next thread blocked on the monitor (if any) to enter the monitor. However, if a monitor is entered more than once, **mon_exit()** returns the previous monitor nesting level without freeing the monitor to other threads. Thus, if the monitor was not reentered, **mon_exit()** returns 0.

**mon_enumerate()** lists all the monitors in the system. The vector supplied is filled in with the ID's of the monitors. *maxsize* is used to avoid exceeding the capacity of the list. If the number of monitors is greater than *maxsize*, only *maxsize* monitor ID's are filled in *vec*.

**mon_waiters()** puts the thread that currently owns the monitor in *owner* and all threads blocked on the monitor in *vec* (subject to the *maxsize* limitation), and returns the number of waiting threads.

**mon_break()** forces the release of a monitor lock not necessarily held by the invoking thread. This enables the next thread blocked on the monitor to enter it.

MONITOR is a macro that can be used at the start of a procedure to indicate that the procedure is a monitor. It uses the exception handling mechanism to ensure that the monitor is exited automatically when the procedure exits. Ordinarily, this single macro replaces paired **mon_enter- mon_exit()** calls in a monitor procedure.

SAMEMON is a convenient predicate used to compare two monitors for equality.

Monitor locks are released automatically when the lwp holding them dies. This may have implications for the validity of the monitor invariant (a condition that is always true *outside* of the monitor) if a thread unexpectedly terminates.

RETURN VALUE

**mon_create()** returns the ID of a new monitor.

A 0 return by **mon_destroy()** indicates success; −1 indicates error.

**mon_enter()** returns the nesting level of the monitor.

Upon successful completion, **mon_exit()** returns the previous nesting level. It returns −1 if there is an error.

**mon_enumerate()** returns the total number of monitors.

**mon_waiters()** returns the number of threads waiting for the monitor.

**mon_cond_enter()** returns the nesting level of the monitor if the monitor is not busy. It return −1 if the monitor is busy.

Upon successful completion, **mon_break()** returns 0. Otherwise, it returns −1.

ERRORS

**mon_destroy()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_INUSE | Attempt to destroy a monitor that has threads blocked on it. |
| LE_NONEXIST | Attempt to destroy non-existent monitor. |

**mon_exit()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_INVALIDARG | Attempt to exit a monitor that the thread does not own. |
| LE_NONEXIST | Attempt to exit non-existent monitor. |

**mon_cond_enter()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_INUSE | The requested monitor is being used by another thread. |
| LE_NONEXIST | Attempt to destroy non-existent monitor. |

**mon_break( )** will fail if one or more of the following are true:

| | |
|---|---|
| LE_NOTOWNED | Attempt to break a monitor lock that is not set. |
| LE_NONEXIST | Attempt to break lock on non-existent monitor. |

**SEE ALSO**

**cv_create**(3L)

**BUGS**

There should be language support to enforce the monitor enter-exit discipline.

NAME

msg_send, msg_recv, msg_reply, MSG_RECVALL, msg_enumsend, msg_enumrecv – LWP send and receive messages

SYNOPSIS

```
#include <lwp/lwp.h>

int
msg_send(dest, arg, argsize, res, ressize)
thread_t dest;    /* destination thread */
caddr_t arg;      /* argument buffer */
int argsize;      /* size of argument buffer */
caddr_t res;      /* result buffer */
int ressize;      /* size of result buffer */

int
msg_recv(sender, arg, argsize, res, ressize, timeout)
thread_t *sender;          /* value-result: sending thread or agent */
caddr_t *arg;              /* argument buffer */
int *argsize;              /* argument size */
caddr_t *res;              /* result buffer */
int *ressize;              /* result size */
struct timeval *timeout;  /* POLL, INFINITY, else timeout */

int
msg_reply(sender)
thread_t sender;/* agent id or thread id */

int
msg_enumsend(vec, maxsize)
thread_t vec[ ];  /* list of blocked senders */
int maxsize;

int
msg_enumrecv(vec, maxsize)
thread_t vec[ ];  /* list of blocked receivers */
int maxsize;

MSG_RECVALL(sender, arg, argsize, res, ressize, timeout)
```

DESCRIPTION

Each thread queues messages addressed to it as they arrive. Threads may either specify that a particular sender's message is to be received next, or that *any* sender's message may be received next.

**msg_send**() specifies a message buffer and a reply buffer, and initiates one half of a rendezvous with the receiver. The sender will block until the receiver replies using **msg_reply**. **msg_recv**() initiates the other half of a rendezvous and blocks the invoking thread until a corresponding **msg_send**() is received. When unblocked by **msg_send**, the receiver may read the message and generate a reply by filling in the reply buffer and issuing **msg_reply**. **msg_reply**() unblocks the sender. Once a reply is sent, the receiver should no longer access either the message or reply buffer.

In **msg_send**, *argsize* specifies the size in bytes of the argument buffer *argbuf*, which is intended to be a read-only (to the receiver) buffer. *ressize* specifies the size in bytes of the result buffer *resbuf*, which is intended to be a write-only (to the receiver) buffer. *dest* is the thread that is the target of the send.

**msg_recv**() blocks the receiver until:

- A message from the agent or thread bound to *sender* has been sent to the receiver or,
- *sender* points to a THREADNULL-valued variable and *any* message has been sent to the receiver from an thread or agent, or,

- After the time specified by *timeout* elapses and no message is received.

If *timeout* is **POLL**, **msg_recv()** returns immediately, returning success if the message expected has arrived; otherwise an error is returned. If *timeout* is **INFINITY**, **msg_recv()** blocks forever or until the expected message arrives. If *timeout* is any other value **msg_recv()** blocks for the time specified by *timeout* or until the expected message arrives, whichever comes first. When **msg_recv()** returns, *sender* is filled in with the identity of the sending thread or agent, and the buffer addresses and sizes specified by the matching send are stored in *arg*, *argsize*, *res*, and *ressize*.

**msg_enumsend()** and **msg_enumrecv()** are used to list all of the threads blocked on sends (awaiting a reply) and receives (awaiting a send), respectively. The value returned is the number of such blocked threads. The vector supplied by the client is filled in (subject to the *maxsize* limitation) with the ID's of the blocked threads. *maxsize* is used to avoid exceeding the capacity of the list. If the number of threads blocked on sends or receives is greater than *maxsize*, only *maxsize* thread ID's are filled in *vec*. If *maxsize* is 0, just the total number of blocked threads is returned.

*sender* in **msg_recv()** is a reference parameter. If you wish to receive from *any* sender, be sure to reinitialize the thread *sender* points to as THREADNULL before each use (do not use the address of THREADNULL for the sender). Alternatively, use the *MSG_RECVALL* macro. This macro has the same parameters that **msg_recv()** does, but will ensure that the sender is properly initialized to allow receipt from any sender. *MSG_RECVALL* returns the result from **msg_recv**.

## RETURN VALUE

Upon successful completion, **msg_send**, **msg_recv()** and **msg_reply()** return 0. Otherwise, -1 is returned.

**msg_enumsend()** returns the number of threads blocked on **msg_send**.

**msg_enumrecv()** returns the number of threads blocked on **msg_recv**.

## ERRORS

**msg_recv()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_TIMEOUT | Timed out before message arrived. |
| LE_INVALIDARG | An illegal timeout was specified or the sender address is that of THREADNULL. |
| LE_NONEXIST | The specified thread or agent does not exist. |

**msg_send()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_INVALIDARG | Attempt to send a message to yourself. |
| LE_NONEXIST | The specified destination thread does not exist or has terminated. |

**msg_reply()** will fail if one or more of the following is true:

| | |
|---|---|
| LE_NOWAIT | Attempt to reply to a sender that is not expecting a reply. |
| LE_NONEXIST | Attempt to reply to a sender that does not exist or has terminated. |

NAME

pod_setmaxpri, pod_getmaxpri, pod_getmaxsize – control LWP scheduling priority

SYNOPSIS

**int**
**pod_setmaxpri(maxprio)**
**int maxprio;**

**int**
**pod_getmaxpri()**

**int**
**pod_getmaxsize()**

DESCRIPTION

The lwp library is self-initializing: the first time you use a primitive that requires threads to be supported, *main* is automatically converted into a thread. A pod will terminate when all client-created lightweight threads (including the thread bound to *main*) are dead.

By default, only a single priority (MINPRIO) is available. However, by using **pod_setmaxpri**, you can make an arbitrary number (up to the limit imposed by the implementation) of priorities available. The *main* thread will receive the highest available scheduling priority at the time of initialization. By using **pod_setmaxpri()** before any other lwp primitives, you can ensure that main will receive the same priority as the argument to **pod_setmaxpri**. **pod_setmaxpri()** can be called repeatedly, as long as the number of scheduling priorities (*maxprio*) increases with each call.

**pod_getmaxpri()** returns the current number of available priorities. Priorities are numbered from 1 (MINPRIO) to maxprio.

The implementation-dependent maximum number of priorities available can be retrieved using **pod_getmaxsize**. This value will never be less than 255.

RETURN VALUE

**pod_setmaxpri()** returns 0 if success; else –1.

**pod_getmaxsize()** returns the maximum number of priorities that your system supports.

**pod_getmaxpri()** returns the number of priority levels set by the most recent **pod_setmaxpri()** call.

ERRORS

**pod_setmaxpri()** will fail if one or more of the following are true:

| | |
|---|---|
| LE_INVALIDARG | Attempt to allocate more priorities than supported. |
| LE_NOROOM | No internal memory left to create pod. |

## NAME

intro – introduction to mathematical library functions and constants

## SYNOPSIS

**#include <sys/ieeefp.h>**

**#include <floatingpoint.h>**

**#include <math.h>**

## DESCRIPTION

The include file **<math.h>** contains declarations of all the functions described in Section 3M that are implemented in the math library, **libm**. C programs should be linked with the the **–lm** option in order to use this library.

**<sys/ieeefp.h>** and **<floatingpoint.h>** define certain types and constants used for **libm** exception handling, conforming to ANSI/IEEE Std 754-1985, the *IEEE Standard for Binary Floating-Point Arithmetic*.

## ACKNOWLEDGEMENT

The Sun version of **libm** is based upon and developed from ideas embodied and codes contained in 4.3 BSD, which may not be compatible with earlier BSD or UNIX implementations.

## IEEE ENVIRONMENT

The IEEE Standard specifies modes for rounding direction, precision, and exception trapping, and status reflecting accrued exceptions. These modes and status constitute the IEEE run-time environment. On Sun-2 and Sun-3 systems without 68881 floating-point co-processors, only the default rounding direction to nearest is available, only the default non-stop exception handling is available, and accrued exception bits are not maintained.

## IEEE EXCEPTION HANDLING

The IEEE Standard specifies exception handling for **aint**, **ceil**, **floor**, **irint**, **remainder**, **rint**, and **sqrt**, and suggests appropriate exception handling for **fp_class**, **copysign**, **fabs**, **finite**, **fmod**, **isinf**, **isnan**, **ilogb**, **ldexp**, **logb**, **nextafter**, **scalb**, **scalbn** and **signbit**, but does not specify exception handling for the other **libm** functions.

For these other unspecified functions the spirit of the IEEE Standard is generally followed in **libm** by handling invalid operand, singularity (division by zero), overflow, and underflow exceptions, as much as possible, in the same way they are handled for the fundamental floating-point operations such as addition and multiplication.

These unspecified functions are usually not quite correctly rounded, may not observe the optional rounding directions, and may not set the inexact exception correctly.

## SYSTEM V EXCEPTION HANDLING

The *System V Interface Definition* (SVID) specifies exception handling for some **libm** functions: **j0()**, **j1()**, **jn()**, **y0()**, **y1()**, **yn()**, **exp()**, **log()**, **log10()**, **pow()**, **sqrt()**, **hypot()**, **lgamma()**, **sinh()**, **cosh()**, **sin()**, **cos()**, **tan()**, **asin()**, **acos()**, and **atan2()**. See **matherr**(3M) for a discussion of the extent to which Sun's implementation of **libm** follows the SVID when it is consistent with the IEEE Standard and with hardware efficiency.

**LIST OF MATH LIBRARY FUNCTIONS**

| Name | Appears on Page | Description |
|---|---|---|
| – | **bessel**(3M) | Bessel functions |
| – | **frexp**(3M) | floating-point analysis |
| – | **hyperbolic**(3M) | hyperbolic functions |
| – | **ieee_functions**(3M) | IEEE classification |
| – | **ieee_test**(3M) | IEEE tests for compliance |
| – | **ieee_values**(3M) | returns double-precision IEEE infinity |
| – | **trig**(3M) | trigonometric functions |
| **acos()** | **trig**(3M) | inverse trigonometric functions |
| **acosh()** | **hyperbolic**(3M) | inverse hyperbolic function |
| **aint()** | **rint**(3M) | convert to integral value in floating-point format |
| **anint()** | **rint**(3M) | convert to integral value in floating-point format |
| **asin()** | **trig**(3M) | inverse trigonometric function |
| **asinh()** | **hyperbolic**(3M) | inverse hyperbolic function |
| **atan()** | **trig**(3M) | inverse trigonometric function |
| **atan2()** | **trig**(3M) | rectangular to polar conversion |
| **atanh()** | **hyperbolic**(3M) | inverse hyperbolic function |
| **cbrt()** | **sqrt**(3M) | cube root |
| **ceil()** | **rint**(3M) | ceiling function |
| **copysign()** | **ieee_functions**(3M) | copy sign bit |
| **cos()** | **trig**(3M) | trigonometric function |
| **cosh()** | **hyperbolic**(3M) | hyperbolic function |
| **erf()** | **erf**(3M) | error function |
| **erfc()** | **erf**(3M) | complementary error function |
| **exp()** | **exp**(3M) | exponential function |
| **expm1()** | **exp**(3M) | exp(X)-1 |
| **exp2()** | **exp**(3M) | 2**X |
| **exp10()** | **exp**(3M) | 10**X |
| **fabs()** | **ieee_functions**(3M) | absolute value function |
| **finite()** | **ieee_functions**(3M) | test for finite number |
| **floor()** | **rint**(3M) | floor function |
| **fmod()** | **ieee_functions**(3M) | floating-point remainder |
| **fp_class()** | **ieee_functions**(3M) | classify operand |
| **frexp()** | **frexp**(3M) | floating-point analysis |
| **hypot()** | **hypot**(3M) | Euclidean distance |
| **ieee_flags()** | **ieee_flags**(3M) | IEEE modes and status |
| **ieee_handler()** | **ieee_handler**(3M) | IEEE trapping |
| **ilogb()** | **ieee_functions**(3M) | exponent extraction |
| **infinity()** | **ieee_values**(3M) | returns double-precision IEEE infinity |
| **irint()** | **rint**(3M) | convert to integral value in integer format |
| **isinf()** | **ieee_functions**(3M) | IEEE classification |
| **isnan()** | **ieee_functions**(3M) | IEEE classification |
| **isnormal()** | **ieee_functions**(3M) | IEEE classification |
| **issubnormal()** | **ieee_functions**(3M) | IEEE classification |
| **iszero()** | **ieee_functions**(3M) | IEEE classification |
| **j0()** | **bessel**(3M) | Bessel function |
| **j1()** | **bessel**(3M) | Bessel function |
| **jn()** | **bessel**(3M) | Bessel function |
| **ldexp()** | **frexp**(3M) | exponent adjustment |
| **lgamma()** | **lgamma**(3M) | log gamma function |
| **log()** | **exp**(3M) | natural logarithm |

| | | |
|---|---|---|
| **logb()** | **ieee_test**(3M) | exponent extraction |
| **log1p()** | **exp**(3M) | log(1+X) |
| **log2()** | **exp**(3M) | log base 2 |
| **log10()** | **exp**(3M) | common logarithm |
| **matherr()** | **matherr**(3M) | math library exception-handling routines |
| **max_normal()** | **ieee_values**(3M) | double-precision IEEE largest positive normalized number |
| **max_subnormal()** | **ieee_values**(3M) | double-precision IEEE largest positive subnormal number |
| **min_normal()** | **ieee_values**(3M) | double-precision IEEE smallest positive normalized number |
| **min_subnormal()** | **ieee_values**(3M) | double-precision IEEE smallest positive subnormal number |
| **modf()** | **frexp**(3M) | floating-point analysis |
| **nextafter()** | **ieee_functions**(3M) | IEEE nearest neighbor |
| **nint()** | **rint**(3M) | convert to integral value in integer format |
| **pow()** | **exp**(3M) | power X**Y |
| **quiet_nan()** | **ieee_values**(3M) | returns double-precision IEEE quiet NaN |
| **remainder()** | **ieee_functions**(3M) | floating-point remainder |
| **rint()** | **rint**(3M) | convert to integral value in floating-point format |
| **scalb()** | **ieee_test**(3M) | exponent adjustment |
| **scalbn()** | **ieee_functions**(3M) | exponent adjustment |
| **signaling_nan()** | **ieee_values**(3M) | returns double-precision IEEE signaling NaN |
| **signbit()** | **ieee_functions**(3M) | IEEE sign bit test |
| **significand()** | **ieee_test**(3M) | scalb(x,-ilogb(x)) |
| **sin()** | **trig**(3M) | trigonometric function |
| **sincos()** | **trig**(3M) | simultaneous sin and cos |
| **single_precision()** | **single_precision**(3M) | single-precision libm access |
| **sinh()** | **hyperbolic**(3M) | hyperbolic function |
| **sqrt()** | **sqrt**(3M) | square root |
| **tan()** | **trig**(3M) | trigonometric function |
| **tanh()** | **hyperbolic**(3M) | hyperbolic function |
| **y0()** | **bessel**(3M) | Bessel function |
| **y1()** | **bessel**(3M) | Bessel function |
| **yn()** | **bessel**(3M) | Bessel function |

**NAME**

j0, j1, jn, y0, y1, yn – Bessel functions

**SYNOPSIS**

```
#include <math.h>

double j0(x)
double x;

double j1(x)
double x;

double jn(n, x)
double x;
int n;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
int n;
```

**DESCRIPTION**

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

**SEE ALSO**

**exp**(3M)

**DIAGNOSTICS**

The functions *y0*, *y1*, and *yn* have logarithmic singularities at the origin, so they treat zero and negative arguments the way *log* does, as described in **exp**(3M). Such arguments are unexceptional for *j0*, *j1*, and *jn*.

**NAME**

erf, erfc – error functions

**SYNOPSIS**

```
#include <math.h>

double erf(x)
double x;

double erfc(x)
double x;
```

**DESCRIPTION**

**erf(x)** returns the error function of $x$; where **erf** (x):= $(2/\sqrt{\pi}) \int_0^x \exp(-t^2)\, dt$.

**erfc(x)** returns 1.0–erf (x), computed however by other methods that avoid cancellation for large $x$.

**NAME**

exp, expm1, exp2, exp10, log, log1p, log2, log10, pow – exponential, logarithm, power

**SYNOPSIS**

```
#include <math.h>

double exp(x)
double x;

double expm1(x)
double x;

double exp2(x)
double x;

double exp10(x)
double x;

double log(x)
double x;

double log1p(x)
double x;

double log2(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;
```

**DESCRIPTION**

**exp**( ) returns the exponential function *e**x*.

**expm1**( ) returns *e**x–1* accurately even for tiny *x*.

**exp2**( ) and **exp10**( ) return *2**x* and *10**x* respectively.

**log**( ) returns the natural logarithm of *x*.

**log1p**( ) returns log(1+x) accurately even for tiny *x*.

**log2**( ) and **log10**( ) return the logarithm to base 2 and 10 respectively.

**pow**( ) returns *x**y*. **pow**(*x*,**0.0**) is 1 for all x, in conformance with 4.3BSD, as discussed in the *Floating Point Programmers Guide*.

**SEE ALSO**

**matherr**(3M)

**DIAGNOSTICS**

All these functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. Thus for x == ±0, **log**(*x*) is −∞ with a division by zero exception; for x < 0, including −∞, **log**(*x*) is a quiet NaN with an invalid operation exception; for x == +∞ or a quiet NaN, **log**(*x*) is x without exception; for x a signaling NaN, **log**(*x*) is a quiet NaN with an invalid operation exception; for x == 1, **log**(*x*) is 0 without exception; for any other positive x, **log**(*x*) is a normalized number with an inexact exception.

In addition, **exp**,**exp2**,**exp10**, **log**,**log2**,**log10**, and **pow**( ) may also set **errno** and call **matherr**(3M).

**NAME**

frexp, modf, ldexp – traditional UNIX functions

**SYNOPSIS**

```
#include <math.h>

double frexp(value, eptr)
double value;
int *eptr;

double ldexp(x,n)
double x;
int n;

double modf(value, iptr)
double value, *iptr;
```

**DESCRIPTION**

These functions are provided for compatibility with other UNIX system implementations. They are not used internally in **libm** or **libc**. Better ways to accomplish similar ends may be found in **ieee_functions**(3M) and **rint**(3M).

**ldexp**(*x*,*n*) returns *x* * *2**n* computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication. Note: **ldexp**(*x*,*n*) differs from **scalbn**(*x*,*n*), defined in **ieee_functions**(3M), only that in the event of IEEE overflow and underflow, **ldexp**(*x*,*n*) sets **errno** to ERANGE.

Every non-zero number can be written uniquely as *x* * *2**n*, where the significand *x* is in the range *0.5* <= */x/* < *1.0* and the exponent *n* is an integer. The function **frexp**( ) returns the significand of a double *value* as a double quantity, *x*, and stores the exponent *n*, indirectly through *eptr*. If *value* == 0, both results returned by **frexp**( ) are 0.

**modf**( ) returns the fractional part of *value* and stores the integral part indirectly through *iptr*. Thus the argument *value* and the returned values **modf**( ) and **iptr* satisfy

(**iptr* + *modf*) == *value*

and both results have the same sign as *value*. The definition of **modf**( ) varies among UNIX system implementations, so avoid **modf**( ) in portable code.

The results of **frexp**( ) and **modf**( ) are not defined when *value* is an IEEE infinity or NaN.

**SEE ALSO**

**ieee_functions**(3M), **rint**(3M)

## NAME

sinh, cosh, tanh, asinh, acosh, atanh – hyperbolic functions

## SYNOPSIS

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;

double asinh(x)
double x;

double acosh(x)
double x;

double atanh(x)
double x;
```

## DESCRIPTION

These functions compute the designated direct and inverse hyperbolic functions for real arguments. They inherit much of their roundoff error from **expm1()** and **log1p**, described in **exp**(3M).

## DIAGNOSTICS

These functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. Thus **sinh()** and **cosh()** return $\pm\infty$ on overflow, **acosh()** returns a NaN if its argument is less than 1, and **atanh()** returns a NaN if its argument has absolute value greater than 1. In addition, **sinh**,**cosh**, and **tanh()** may also set **errno** and call **matherr**(3M).

## SEE ALSO

**exp**(3M), **matherr**(3M)

NAME

hypot – Euclidean distance

SYNOPSIS

**#include <math.h>**

**double hypot(x, y)**
**double x, y;**

DESCRIPTION

**hypot( )** returns

**sqrt(x*x + y*y) ,**

taking precautions against unwarranted IEEE exceptions. On IEEE overflow, **hypot( )** may also set **errno** and call **matherr**(3M). **hypot(**$\pm\infty$**, y)** is $+\infty$ for any y, even a NaN, and is exceptional only for a signaling NaN.

**hypot(x,y)** and **atan2**(3M) convert rectangular coordinates $(x,y)$ to polar $(r,\theta)$; **hypot( )** computes $r$, the modulus or radius.

SEE ALSO

**matherr**(3M)

**NAME**

ieee_flags – mode and status function for IEEE standard arithmetic

**SYNOPSIS**

**#include <sys/ieeefp.h>**

**int ieee_flags(action,mode,in,out)**
**char *action, *mode, *in, **out;**

**DESCRIPTION**

This function provides easy access to the modes and status required to fully exploit ANSI/IEEE Std 754-1985 arithmetic in a C program. All arguments are pointers to strings. Results arising from invalid arguments and invalid combinations are undefined for efficiency.

There are four types of *action*: "get", "set", "clear", and "clearall". There are three valid settings for *mode*, two corresponding to modes of IEEE arithmetic:

| | |
|---|---|
| "direction", | ... current rounding direction mode |
| "precision", | ... current rounding precision mode |

and one corresponding to status of IEEE arithmetic:

| | |
|---|---|
| "exception". | ... accrued exception-occurred status |

There are 14 types of *in* and *out :*

| | |
|---|---|
| "nearest", | ... round toward nearest |
| "tozero", | ... round toward zero |
| "negative", | ... round toward negative infinity |
| "positive", | ... round toward positive infinity |
| "extended", | |
| "double", | |
| "single", | |
| "inexact", | |
| "division", | ... division by zero exception |
| "underflow", | |
| "overflow", | |
| "invalid", | |
| "all", | ... all five exceptions above |
| "common". | ... invalid, overflow, and division exceptions |

Note: "all" and "common" only make sense with "set" or "clear".

For "clearall", **ieee_flags()** returns 0 and restores all default modes and status. Nothing will be assigned to *out*. Thus

```
char *mode, *out, *in;
ieee_flags("clearall",mode, in, &out);
```

set rounding direction to "nearest", rounding precision to "extended", and all accrued exception-occurred status to zero.

For "clear", **ieee_flags()** returns 0 and restores the default mode or status. Nothing will be assigned to *out*. Thus

```
char *out, *in;
ieee_flags("clear","direction", in, &out);      ... set rounding direction to round to nearest.
```

For "set", **ieee_flags()** returns 0 if the action is successful and 1 if the corresponding required status or mode is not available (for instance, not supported in hardware). Nothing will be assigned to *out*. Thus

```
char *out, *in;
ieee_flags ("set","direction","tozero",&out);      ... set rounding direction to round toward zero;
```

For "get", we have the following cases:

Case 1: *mode* is "direction". In that case, *out* returns one of the four strings "nearest", "tozero", "positive", "negative"; and **ieee_flags()** returns a value corresponding to *out* according to the enum *fp_direction_type* defined in **<sys/ieeefp.h>**.

Case 2: *mode* is "precision". In that case, *out* returns one of the three strings "extended", "double", "single"; and **ieee_flags()** returns a value corresponding to *out* according to the enum *fp_precision_type* defined in **<sys/ieeefp.h>**.

Case 3: *mode* is "exception". In that case, *out* returns

**(a)** "not available" if information on exception is not available,
**(b)** "no exception" if no accrued exception,
**(c)** the accrued exception that has the highest priority according to the list below

**(1)** the exception named by *in*,
**(2)** "invalid",
**(3)** "overflow",
**(4)** "division",
**(5)** "underflow",
**(6)** "inexact".

In this case **ieee_flags()** returns a five bit value where each bit (cf. enum *fp_exception_type* in **<sys/ieeefp.h>**) corresponds to an exception-occurred accrued status flag: 0 = off, 1 = on. The bit corresponding to a particular exception varies among architectures.

Example:

```
char *out; int k, ieee_flags();
ieee_flags ("clear","exception","all",&out);      /* clear all accrued exceptions */
...
... (code that generates three exceptions: overflow, invalid, inexact)
...
k = ieee_flags("get","exception","overflow",&out);
```

then out = "overflow", and on a Sun-3, k=25.

FILES

**/usr/include/sys/ieeefp.h**
**/usr/lib/libm.a**

NAME

ieee_functions, fp_class, finite, ilogb, isinf, isnan, isnormal, issubnormal, iszero, signbit, copysign, fabs, fmod, nextafter, remainder, scalbn – appendix and related miscellaneous functions for IEEE arithmetic

SYNOPSIS

```
#include <math.h>

enum fp_class_type fp_class(x)
double x;

int finite(x)
double x;

int ilogb(x)
double x;

int isinf(x)
double x;

int isnan(x)
double x;

int isnormal(x)
double x;

int issubnormal(x)
double x;

int iszero(x)
double x;

int signbit(x)
double x;

double copysign(x,y)
double x, y;

double fabs(x)
double x;

double fmod(x,y)
double x, y;

double nextafter(x,y)
double x, y;

double remainder(x,y)
double x, y;

double scalbn(x,n)
double x; int n;
```

## DESCRIPTION

Most of these functions provide capabilities required by ANSI/IEEE Std 754-1985 or suggested in its appendix.

**fp_class**($x$) corresponds to the IEEE's class() and classifies x as zero, subnormal, normal, ∞, or quiet or signaling *NaN*; **<floatingpoint.h>** defines *enum fp_class_type*. The following functions return 0 if the indicated condition is not satisfied:

| | |
|---|---|
| **finite**($x$) | returns 1 if x is zero, subnormal or normal |
| **isinf**($x$) | returns 1 if $x$ is ∞ |
| **isnan**($x$) | returns 1 if $x$ is *NaN* |
| **isnormal**($x$) | returns 1 if $x$ is normal |
| **issubnormal**($x$) | returns 1 if $x$ is subnormal |
| **iszero**($x$) | returns 1 if $x$ is zero |
| **signbit**($x$) | returns 1 if $x$'s sign bit is set |

**ilogb**($x$) returns the unbiased exponent of $x$ in integer format. **ilogb**(±∞) = +MAXINT and **ilogb(0)** = −MAXINT; **<values.h>** defines MAXINT as the largest int. **ilogb**($x$) never generates an exception. When $x$ is subnormal, **ilogb**($x$) returns an exponent computed as if $x$ were first normalized.

**copysign**($x$,$y$) returns $x$ with $y$'s sign bit.

**fabs**($x$) returns the absolute value of $x$.

**nextafter**($x$,$y$) returns the next machine representable number from $x$ in the direction $y$.

**remainder**($x$,$y$) and **fmod**($x$,$y$) return a remainder of $x$ with respect to $y$; that is, the result $r$ is one of the numbers that differ from $x$ by an integral multiple of $y$. Thus (x-r)/y is an integral value, even though it might exceed MAXINT if it were explicitly computed as an int. Both functions return one of the two such r smallest in magnitude. **remainder**($x$,$y$) is the operation specified in ANSI/IEEE Std 754-1985; the result of **fmod**($x$,$y$) may differ from **remainder**'$s$ result by ±$y$. The magnitude of **remainder**'$s$ result can not exceed half that of $y$; its sign might not agree with either $x$ or $y$. The magnitude of **fmod**'$s$ result is less than that of $y$; its sign agrees with that of $x$. Neither function can generate an exception as long as both arguments are normal or subnormal. **remainder( x , 0), fmod( x , 0), remainder(∞, y )**, and **fmod(∞, y )** are invalid operations that produce a *NaN*.

**scalbn**($x$,$n$) returns $x$* **2****n** computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication. Thus

**1 ≤ scalbn(fabs(**$x$**),−ilogb(**$x$**)) < 2**

for every $x$ except 0, ∞, and *NaN*.

## FILES

**/usr/include/floatingpoint.h**
**/usr/include/math.h**
**/usr/include/values.h**
**/usr/lib/libm.a**

## SEE ALSO

**floatingpoint**(3), **ieee_environment**(3M), **matherr**(3M)

## NAME

ieee_handler – IEEE exception trap handler function

## SYNOPSIS

**#include <floatingpoint.h>**

**int ieee_handler(action,exception,hdl)**
**char action[ ], exception[ ];**
**sigfpe_handler_type hdl;**

## DESCRIPTION

This function provides easy exception handling to exploit ANSI/IEEE Std 754-1985 arithmetic in a C program. All arguments are pointers to strings. Results arising from invalid arguments and invalid combinations are undefined for efficiency.

There are three types of *action :* "get", "set", and "clear". There are five types of *exception :*

| | |
|---|---|
| "inexact" | |
| "division" | ... division by zero exception |
| "underflow" | |
| "overflow" | |
| "invalid" | |
| "all" | ... all five exceptions above |
| "common" | ... invalid, overflow, and division exceptions |

Note: "all" and "common" only make sense with "set" or "clear".

**hdl** contains the address of a signal-handling routine. **<floatingpoint.h>** defines *sigfpe_handler_type*.

"get" will get the location of the current handler routine for *exception* in **hdl** . "set" will set the routine pointed at by **hdl** to be the handler routine and at the same time enable the trap on *exception*, except when **hdl** == **SIGFPE_DEFAULT** or **SIGFPE_IGNORE**; then **ieee_handler( )** will disable the trap on *exception*. When **hdl** == **SIGFPE_ABORT**, any trap on *exception* will dump core using **abort**(3). "clear" "all" disables trapping on all five exceptions.

Two steps are required to intercept an IEEE-related **SIGFPE** code with **ieee_handler**:

1) Set up a handler with **ieee_handler**.

2) Perform a floating-point operation that generates the intended IEEE exception.

Unlike **sigfpe**(3), **ieee_handler( )** also adjusts floating-point hardware mode bits affecting IEEE trapping. For "clear", "set" **SIGFPE_DEFAULT**, or "set" **SIGFPE_IGNORE**, the hardware trap is disabled. For any other "set", the hardware trap is enabled.

**SIGFPE** signals can be handled using **sigvec**(2), **signal**(3), **signal**(3F), **sigfpe**(3), or **ieee_handler**(3M). In a particular program, to avoid confusion, use only one of these interfaces to handle **SIGFPE** signals.

## DIAGNOSTICS

**ieee_handler( )** normally returns 0. In the case of "set", 1 will be returned if the action is not available (for instance, not supported in hardware).

**EXAMPLE**

A user-specified signal handler might look like this:

```
void sample_handler( sig, code, scp, addr)
int sig ;          /* sig == SIGFPE always */
int code ;
struct sigcontext *scp ;
char *addr ;
{
        /*
          Sample user-written sigfpe code handler.
          Prints a message and continues.
          struct sigcontext is defined in <signal.h>.
         */
        printf("ieee exception code %x occurred at pc %X \n",code,scp->sc_pc);
}
```

and it might be set up like this:

```
extern void sample_handler();
main()
{
        sigfpe_handler_type hdl, old_handler1, old_handler2;
/*
* save current overflow and invalid handlers
*/
        ieee_handler("get","overflow",old_handler1);
        ieee_handler("get","invalid", old_handler2);
/*
* set new overflow handler to sample_handler() and set new
* invalid handler to SIGFPE_ABORT (abort on invalid)
*/
        hdl = (sigfpe_handler_type) sample_handler;
        if(ieee_handler("set","overflow",hdl) != 0)
                printf("ieee_handler can't set overflow \n");
        if(ieee_handler("set","invalid",SIGFPE_ABORT) != 0)
                printf("ieee_handler can't set invalid \n");
        ...
/*
* restore old overflow and invalid handlers
*/
        ieee_handler("set","overflow", old_handler1);
        ieee_handler("set","invalid", old_handler2);
}
```

**FILES**

**/usr/include/floatingpoint.h**
**/usr/include/signal.h**
**/usr/lib/libm.a**

**SEE ALSO**

**sigvec**(2), **abort**(3), **floatingpoint**(3), **sigfpe**(3), **signal**(3), **signal**(3F)

## NAME

ieee_test, logb, scalb, significand – IEEE test functions for verifying standard compliance

## SYNOPSIS

```
#include <math.h>

double logb(x)
double x;

double scalb(x,y)
double x; double y;

double significand(x)
double x;
```

## DESCRIPTION

These functions allow users to verify compliance to ANSI/IEEE Std 754-1985 by running certain test vectors distributed by the University of California. Their use is not otherwise recommended; instead use **scalbn**(*x*,*n*) and **ilogb**(*x*) described in **ieee_functions**(3M). See the *Floating Point Programmers Guide* for details.

**logb**(*x*) returns the unbiased exponent of *x* in floating-point format, for exercising the logb(L) test vector. **logb**(±∞) = +∞; **logb(0)** = −∞ with a division by zero exception. **logb**(*x*) differs from **ilogb**(*x*) in returning a result in floating-point rather than integer format, in sometimes signaling IEEE exceptions, and in not normalizing subnormal *x*.

**scalb**(*x*,**(double)n**) returns *x* * *2**n* computed by exponent manipulation rather than by actually performing an exponentiation or a multiplication, for exercising the scalb(S) test vector. Thus

**0 ≤ scalb(fabs**(*x*),−**logb**(*x*)) < **2**

for every *x* except 0, ∞ and *NaN*. **scalb**(*x*,*y*) is not defined when *y* is not an integral value. **scalb**(*x*,*y*) differs from **scalbn**(*x*,**n**) in that the second argument is in floating-point rather than integer format.

**significand**(*x*) computes just

**scalb**(*x*, **(double) -ilogb**(*x*)),

for exercising the fraction-part(F) test vector.

## FILES

**/usr/include/math.h**
**/usr/lib/libm.a**

## SEE ALSO

**floatingpoint**(3), **ieee_values**(3M), **ieee_functions**(3M), **matherr**(3M)

**NAME**

ieee_values, min_subnormal, max_subnormal, min_normal, max_normal, infinity, quiet_nan, signaling_nan, HUGE, HUGE_VAL – functions that return extreme values of IEEE arithmetic

**SYNOPSIS**

```
#include <math.h>

double min_subnormal()

double max_subnormal()

double min_normal()

double max_normal()

double infinity()

double quiet_nan(n)
long n;

double signaling_nan(n)
long n;

#define HUGE (infinity())

#define HUGE_VAL (infinity())
```

**DESCRIPTION**

These functions return special values associated with ANSI/IEEE Std 754-1985 double-precision floating-point arithmetic: the smallest and largest positive subnormal numbers, the smallest and largest positive normalized numbers, positive infinity, and a quiet and signaling NaN. The long parameters *n* to **quiet_nan(***n***)** and **signaling_nan(***n***)** are presently unused but are reserved for future use to specify the significand of the returned NaN.

None of these functions are affected by IEEE rounding or trapping modes or generate any IEEE exceptions.

The macro **HUGE** returns $+\infty$ in accordance with previous SunOS releases. The macro **HUGE_VAL** returns $+\infty$ in accordance with the System V Interface Definition.

**FILES**

**/usr/include/math.h**
**/usr/lib/libm.a**

**SEE ALSO**

**ieee_functions**(3M)

**NAME**

lgamma – log gamma function

**SYNOPSIS**

```
#include <math.h>

extern int signgam;

double lgamma(x)
double x;
```

**DESCRIPTION**

**lgamma( )** returns

$$\ln|\Gamma(x)|$$

where

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$$

for $x > 0$ and

$$\Gamma(x) = \pi/(\Gamma(1-x)\sin(\pi x))$$

for $x < 1$.

The external integer **signgam** returns the sign of $\Gamma(x)$.

**IDIOSYNCRASIES**

Do *not* use the expression **signgam*exp(lgamma(x))** to compute '**g := Γ(x)**'. Instead compute **lgamma( )** first:

```
lg = lgamma(x); g = signgam*exp(lg);
```

only after **lgamma( )** has returned can **signgam** be correct. Note: $\Gamma(x)$ must overflow when $x$ is large enough, underflow when $-x$ is large enough, and generate a division by zero exception at the singularities $x$ a nonpositive integer. In addition, **lgamma( )** may also set **errno** and call **matherr**(3M).

**SEE ALSO**

**matherr**(3M)

## NAME

matherr – math library exception-handling function

## SYNOPSIS

```
#include <math.h>

int matherr(exc)
struct exception *exc;
```

## DESCRIPTION

The SVID (*System V Interface Definition*) specifies that certain **libm** functions call **matherr( )** when exceptions are detected. Users may define their own mechanisms for handling exceptions, by including a function named **matherr( )** in their programs. **matherr( )** is of the form described above. When an exception occurs, a pointer to the exception structure *exc* will be passed to the user-supplied **matherr( )** function. This structure, which is defined in the **<math.h>** header file, is as follows:

```
struct exception {
        int type;
        char *name;
        double arg1, arg2, retval;
};
```

The element **type** is an integer describing the type of exception that has occurred, from the following list of constants (defined in the header file):

| | |
|---|---|
| **DOMAIN** | argument domain exception |
| **SING** | argument singularity |
| **OVERFLOW** | overflow range exception |
| **UNDERFLOW** | underflow range exception |

The element **name** points to a string containing the name of the function that incurred the exception. The elements **arg1** and **arg2** are the arguments with which the function was invoked. **retval** is set to the default value that will be returned by the function unless the user's **matherr( )** sets it to a different value.

If the user's **matherr( )** function returns non-zero, no exception message will be printed, and **errno** will not be set.

If **matherr( )** is not supplied by the user, the default matherr exception-handling mechanisms, summarized in the table below, will be invoked upon exception:

**DOMAIN==fp_invalid**
An IEEE NaN is usually returned, **errno** is set to EDOM, and a message is printed on standard error. **pow(*x*,0.0)** for any *x* and **atan2(0.0,0.0)** return numerical default results but set **errno** and print the message.

**SING==fp_division**
An IEEE ∞ of appropriate sign is returned, **errno** is set to EDOM, and a message is printed on standard error.

**OVERFLOW==fp_overflow**
In the default rounding direction, an IEEE ∞ of appropriate sign is returned. In optional rounding directions, ±**MAXDOUBLE**, the largest finite double-precision number, is sometimes returned instead of ±∞. **errno** is set to ERANGE.

**UNDERFLOW==fp_underflow**
An appropriately-signed zero, subnormal number, or smallest normalized number is returned, and **errno** is set to ERANGE.

The facilities provided by **matherr( )** are not available in situations such as compiling on a Sun-3 system with **/usr/lib/f68881/libm.il** or **/usr/lib/ffpa/libm.il**, in which case some **libm** functions are converted to atomic hardware operations. In these cases setting **errno** and calling **matherr( )** are not worth the adverse performance impact, but regular ANSI/IEEE Std 754-1985 exception handling remains available. In any

case **errno** is not a reliable error indicator in that it may be unexpectedly set by a function in a handler for an asynchronous signal.

**DEFAULT ERROR HANDLING PROCEDURES**

| | *Types of Errors* | | | |
|---|---|---|---|---|
| <math.h> type | DOMAIN | SING | OVERFLOW | UNDERFLOW |
| **errno** | EDOM | EDOM | ERANGE | ERANGE |
| IEEE Exception | Invalid Operation | Division by Zero | Overflow | Underflow |
| <floatingpoint.h> type | fp_invalid | fp_division | fp_overflow | fp_underflow |
| ACOS, ASIN: | M, NaN | – | – | – |
| ATAN2(0,0): | M, ±0.0 or ±π | – | – | – |
| BESSEL: | | | | |
| y0, y1, yn (x < 0) | M, NaN | – | – | – |
| y0, y1, yn (x = 0) | – | M, −∞ | – | – |
| COSH, SINH: | – | – | IEEE Overflow | – |
| EXP: | – | – | IEEE Overflow | IEEE Underflow |
| HYPOT: | – | – | IEEE Overflow | – |
| LGAMMA: | – | M, +∞ | IEEE Overflow | – |
| LOG, LOG10: | | | | |
| (x < 0) | M, NaN | – | – | – |
| (x = 0) | – | M, −∞ | – | – |
| POW: | | | | |
| usual cases | – | – | IEEE Overflow | IEEE Underflow |
| (x < 0) ** (y not an integer) | M, NaN | – | – | – |
| 0 ** 0 | M, 1.0 | – | – | – |
| 0 ** (y < 0) | – | M, ±∞ | – | – |
| SQRT: | M, NaN | – | – | – |

**ABBREVIATIONS**

| | |
|---|---|
| M | Message is printed (EDOM exception). |
| NaN | IEEE NaN result and invalid operation exception. |
| ∞ | IEEE ∞ result and division-by-zero exception. |
| IEEE Overflow | IEEE Overflow result and exception. |
| IEEE Underflow | IEEE Underflow result and exception. |
| π | Closest machine-representable approximation to **pi**. |

The interaction of IEEE arithmetic and **matherr( )** is not defined when executing under IEEE rounding modes other than the default round to nearest: **matherr( )** may not be called on overflow or underflow, and the Sun-provided **matherr( )** may return results that differ from those in this table.

**EXAMPLE**

```
#include <math.h>

int
matherr(x)
register struct exception *x;
{
        switch (x->type) {
        case
                DOMAIN:
                /* change sqrt to return sqrt(-arg1), not NaN */
                if (!strcmp(x->name, "sqrt")) {
                        x->retval = sqrt(-x->arg1);
                        return (0); /* print message and set errno */
        } /* fall through */
        case
                SING:
                /* all other domain or sing exceptions, print message and abort */
                fprintf(stderr, "domain exception in %s\n", x->name);
                abort( );
                break;
        }
        return (0); /* all other exceptions, execute default procedure */
}
```

NAME

aint, anint, ceil, floor, rint, irint, nint – round to integral value in floating-point or integer format

SYNOPSIS

```
#include <math.h>

double aint(x)
double x;

double anint(x)
double x;

double ceil(x)
double x;

double floor(x)
double x;

double rint(x)
double x;

int irint(x)
double x;

int nint(x)
double x;
```

DESCRIPTION

**aint**, **anint**, **ceil**, **floor**, and **rint**() convert a double value into an integral value in double format. They vary in how they choose the result when the argument is not already an integral value. Here an "integral value" means a value of a mathematical integer, which however might be too large to fit in a particular computer's int format. All sufficiently large values in a particular floating-point format are already integral; in IEEE double-precision format, that means all values >= 2**52. Zeros, infinities, and quiet NaNs are treated as integral values by these functions, which always preserve their argument's sign.

**aint**() returns the integral value between $x$ and 0, nearest $x$. This corresponds to IEEE rounding toward zero and to the Fortran generic intrinsic function **aint**.

**anint**() returns the nearest integral value to $x$, except halfway cases are rounded to the integral value larger in magnitude. This corresponds to the Fortran generic intrinsic function **anint**.

**ceil**() returns the least integral value greater than or equal to $x$. This corresponds to IEEE rounding toward positive infinity.

**floor**() returns the greatest integral value less than or equal to $x$. This corresponds to IEEE rounding toward negative infinity.

**rint**() rounds $x$ to an integral value according to the current IEEE rounding direction.

**irint** converts $x$ into int format according to the current IEEE rounding direction.

**nint**() converts $x$ into int format rounding to the nearest int value, except halfway cases are rounded to the int value larger in magnitude. This corresponds to the Fortran generic intrinsic function **nint**.

NAME

single_precision - Single-precision access to math library functions

SYNOPSIS

**#include <math.h>**

**FLOATFUNCTIONTYPE r_acos_ (x)**
**FLOATFUNCTIONTYPE r_acosh_ (x)**
**FLOATFUNCTIONTYPE r_aint_ (x)**
**FLOATFUNCTIONTYPE r_anint_ (x)**
**FLOATFUNCTIONTYPE r_asin_ (x)**
**FLOATFUNCTIONTYPE r_asinh_ (x)**
**FLOATFUNCTIONTYPE r_atan_ (x)**
**FLOATFUNCTIONTYPE r_atanh_ (x)**
**FLOATFUNCTIONTYPE r_atan2_ (x,y)**
**FLOATFUNCTIONTYPE r_cbrt_ (x)**
**FLOATFUNCTIONTYPE r_ceil_ (x)**
**enum fp_class_type ir_fp_class_ (x)**
**FLOATFUNCTIONTYPE r_copysign_ (x,y)**
**FLOATFUNCTIONTYPE r_cos_ (x)**
**FLOATFUNCTIONTYPE r_cosh_ (x)**
**FLOATFUNCTIONTYPE r_erf_ (x)**
**FLOATFUNCTIONTYPE r_erfc_ (x)**
**FLOATFUNCTIONTYPE r_exp_ (x)**
**FLOATFUNCTIONTYPE r_expm1_ (x)**
**FLOATFUNCTIONTYPE r_exp2_ (x)**
**FLOATFUNCTIONTYPE r_exp10_ (x)**
**FLOATFUNCTIONTYPE r_fabs_ (x)**
**int ir_finite_ (x)**
**FLOATFUNCTIONTYPE r_floor_ (x)**
**FLOATFUNCTIONTYPE r_fmod_ (x,y)**
**FLOATFUNCTIONTYPE r_hypot_ (x,y)**
**int ir_ilogb_ (x)**
**int ir_irint_ (x)**
**int ir_isinf_ (x)**
**int ir_isnan_ (x)**
**int ir_isnormal_ (x)**
**int ir_issubnormal_ (x)**
**int ir_iszero_ (x)**
**int ir_nint_ (x)**
**FLOATFUNCTIONTYPE r_infinity_ ( )**
**FLOATFUNCTIONTYPE r_j0_ (x)**
**FLOATFUNCTIONTYPE r_j1_ (x)**
**FLOATFUNCTIONTYPE r_jn_ (n,x)**
**FLOATFUNCTIONTYPE r_lgamma_ (x)**
**FLOATFUNCTIONTYPE r_logb_ (x)**
**FLOATFUNCTIONTYPE r_log_ (x)**
**FLOATFUNCTIONTYPE r_log1p_ (x)**
**FLOATFUNCTIONTYPE r_log2_ (x)**
**FLOATFUNCTIONTYPE r_log10_ (x)**
**FLOATFUNCTIONTYPE r_max_normal_ ( )**
**FLOATFUNCTIONTYPE r_max_subnormal_ ( )**
**FLOATFUNCTIONTYPE r_min_normal_ ( )**
**FLOATFUNCTIONTYPE r_min_subnormal_ ( )**
**FLOATFUNCTIONTYPE r_nextafter_ (x,y)**

**FLOATFUNCTIONTYPE r_pow_ (x,y)**
**FLOATFUNCTIONTYPE r_quiet_nan_ (n)**
**FLOATFUNCTIONTYPE r_remainder_ (x,y)**
**FLOATFUNCTIONTYPE r_rint_ (x)**
**FLOATFUNCTIONTYPE r_scalb_ (x,y)**
**FLOATFUNCTIONTYPE r_scalbn_ (x,n)**
**FLOATFUNCTIONTYPE r_signaling_nan_ (n)**
**int ir_signbit_ (x)**
**FLOATFUNCTIONTYPE r_significand_ (x)**
**FLOATFUNCTIONTYPE r_sin_ (x)**
**void r_sincos_ (x,s,c)**
**FLOATFUNCTIONTYPE r_sinh_ (x)**
**FLOATFUNCTIONTYPE r_sqrt_ (x)**
**FLOATFUNCTIONTYPE r_tan_ (x)**
**FLOATFUNCTIONTYPE r_tanh_ (x)**
**FLOATFUNCTIONTYPE r_y0_ (x)**
**FLOATFUNCTIONTYPE r_y1_ (x)**
**FLOATFUNCTIONTYPE r_yn_ (n,x)**

**float *x, *y, *s, *c**
**int *n**

## DESCRIPTION

These functions are single-precision versions of certain **libm** functions. Primarily for use by Fortran programmers, these functions may also be used in other languages. The single-precision floating-point results are deviously declared to avoid C's automatic type conversion to double.

## FILES

**/usr/lib/libm.a**

NAME

sqrt, cbrt – cube root, square root

SYNOPSIS

```
#include <math.h>

double cbrt(x)
double x;

double sqrt(x)
double x;
```

DESCRIPTION

**sqrt**(*x*) returns the square root of *x*, correctly rounded according to ANSI/IEEE 754-1985. In addition, **sqrt**( ) may also set **errno** and call **matherr**(3M).

**cbrt**(*x*) returns the cube root of *x*. **cbrt**( ) is accurate to within 0.7 *ulps*.

SEE ALSO

**matherr**(3M)

**NAME**

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

**SYNOPSIS**

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

void sincos(x, s, c)
double x, *s, *c;

double tan(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(y, x)
double y, x;
```

**DESCRIPTION**

**sin, cos, sincos,** and **tan( )** return trigonometric functions of radian arguments. The values of trigonometric functions of arguments exceeding $\pi/4$ in magnitude are affected by the precision of the approximation to $\pi/2$ used to reduce those arguments to the range $-\pi/4$ to $\pi/4$. Argument reduction may occur in hardware or software; if in software, the variable **fp_pi** defined in **<math.h>** allows changing that precision at run time. Trigonometric argument reduction is discussed in the *Floating Point Programmers Guide*. Note that **sincos(x,s,c)** allows simultaneous computation of ***s** = **sin(x)** and ***c** = **cos(x)**.

**asin( )** returns the arc sin in the range $-\pi/2$ to $\pi/2$.

**acos( )** returns the arc cosine in the range 0 to $\pi$.

**atan( )** returns the arc tangent of $x$ in the range $-\pi/2$ to $\pi/2$.

**atan2**($y$,$x$) and **hypot**(3M) convert rectangular coordinates ($x$,$y$) to polar ($r$,$\theta$); **atan2( )** computes $\theta$, the argument or phase, by computing an arc tangent of $y/x$ in the range $-\pi$ to $\pi$. **atan2**(0.0,0.0) is $\pm 0.0$ or $\pm\pi$, in conformance with 4.3BSD, as discussed in the *Floating Point Programmers Guide*.

**DIAGNOSTICS**

These functions handle exceptional arguments in the spirit of ANSI/IEEE Std 754-1985. **sin**($\pm\infty$), **cos**($\pm\infty$), **tan**($\pm\infty$), or **asin**($x$) or **acos**($x$) with $|x|>1$, return NaN. In addition, **asin**, **acos**, and **atan2( )** may also set **errno** and call **matherr**(3M).

**SEE ALSO**

**hypot**(3M), **matherr**(3M)

## NAME

intro – introduction to RPC service library functions and protocols

## DESCRIPTION

These functions constitute the RPC service library. Most of these describe RPC protocols. The PROTOCOL section describes how to access the protocol description file. This file may be compiled with **rpcgen**(1) to produce data definitions and XDR routines. Procompiled versions of header files sometimes exist as **<rpcsvc/*.h>** and precompiled XDR routines and programming interfaces to the protocols sometimes exist in *librpcsvc*. Warning: some of these header files and XDR routines were hand-written because they existed before *rpcgen*. They do not correspond to their protocol description file. In order to get the link editor to load this library, use the **–lrpcsvc** option of **cc**(1V). Information about the availability of programming interfaces to these protocols is available under PROGRAMMING section of each manual page.

Some routines in the **librpcsvc** library do not correspond to protocols, but are useful utilities for RPC programming. These are distinguished by the presence of the SYNOPSIS section instead of the usual PROTOCOL section.

## LIST OF STANDARD RPC SERVICES

| Name | Appears on Page | Description |
|---|---|---|
| **bootparam()** | **bootparam**(3R) | bootparam protocol |
| **ether()** | **ether**(3R) | monitor traffic on the Ethernet |
| **get()** | **publickey**(3R) | get secret key |
| **getrpcport()** | **getrpcport**(3R) | get RPC port number |
| **getsecretkey()** | **publickey**(3R) | get secret key |
| **ipalloc()** | **ipalloc**(3R) | determine or temporarily allocate IP address |
| **key()** | **publickey**(3R) | get secret key |
| **klm_prot()** | **klm_prot**(3R) | protocol between kernel and local lock manager |
| **mount()** | **mount**(3R) | keep track of remotely mounted filesystems |
| **nlm_prot()** | **nlm_prot**(3R) | protocol between local and remote network lock managers |
| **pnp()** | **pnp**(3R) | Automated network installer |
| **public()** | **publickey**(3R) | get secret key |
| **rex()** | **rex**(3R) | remote execution protocol |
| **rnusers()** | **rnusers**(3R) | return information about users on remote machines |
| **rquota()** | **rquota**(3R) | implement quotas on remote machines |
| **rstat()** | **rstat**(3R) | get performance data from remote kernel |
| **rusers()** | **rnusers**(3R) | return information about users on remote machines |
| **rwall()** | **rwall**(3R) | write to specified remote machines |
| **secret()** | **publickey**(3R) | get secret key |
| **sm_inter()** | **sm_inter**(3R) | status monitor protocol |
| **spray()** | **spray**(3R) | scatter data in order to check the network |
| **xcrypt()** | **xcrypt**(3R) | hex encryption and utility routines |
| **yp()** | **yp**(3R) | Yellow Pages protocol |
| **yppasswd()** | **yppasswd**(3R) | update user password in Yellow Pages |

**NAME**

bootparam – bootparam protocol

**PROTOCOL**

**/usr/include/rpcsvc/bootparam_prot.x**

**DESCRIPTION**

The bootparam protocol is used for providing information to the diskless clients necessary for booting.

**PROGRAMMING**

**#include <rpcsvc/bootparam.h>**

**XDR Routines**

The following XDR routines are available in **librpcsvc**:

**xdr_bp_whoami_arg**
**xdr_bp_whoami_res**
**xdr_bp_getfile_arg**
**xdr_bp_getfile_res**

**SEE ALSO**

**bootparams**(5), **bootparamd**(8)

**NAME**

ether – monitor traffic on the Ethernet

**PROTOCOL**

**/usr/include/rpcsvc/ether.x**

**DESCRIPTION**

The ether protocol is used for monitoring traffic on the ethernet.

**PROGRAMMING**

**#include <rpcsvc/ether.h>**

The following XDR routines are available in **librpcsvc**:

**xdr_etherstat**
**xdr_etheraddrs**
**xdr_etherhtable**
**xdr_etherhmem**
**xdr_addrmask**

**SEE ALSO**

**traffic**(1C), **etherfind**(8C), **etherd**(8C)

**NAME**

getrpcport – get RPC port number

**SYNOPSIS**

```
int getrpcport(host, prognum, versnum, proto)
        char *host;
        int prognum, versnum, proto;
```

**DESCRIPTION**

**getrpcport()** returns the port number for version *versnum* of the RPC program *prognum* running on *host* and using protocol *proto*. It returns 0 if it cannot contact the portmapper, or if *prognum* is not registered. If *prognum* is registered but not with version *versnum*, it will still return a port number (for some version of the program) indicating that the program is indeed registered. The version mismatch will be detected upon the first call to the service.

## NAME

klm_prot – protocol between kernel and local lock manager

## PROTOCOL

**/usr/include/rpcsvc/klm_prot.x**

## DESCRIPTION

The protocol is used for communication between kernel and local lock manager.

## PROGRAMMING

**#include <rpcsvc/klm_prot.h>**

### XDR Routines

The following XDR routines are available in **librpcsvc**:

**xdr_klm_testargs**
**xdr_klm_testrply**
**xdr_klm_lockargs**
**xdr_klm_unlockargs**
**xdr_klm_stat**

## SEE ALSO

**lockd**(8C)

## NAME

mount – keep track of remotely mounted filesystems

## PROTOCOL

**/usr/include/rpcsvc/mount.x**

## DESCRIPTION

The mount protocol is separate from, but related to, the NFS protocol. It provides all of the operating system specific servies to get the NFS off the ground — looking up path names, validating user identity, and checking access permissions. Clients use the mount protocol to get the first file handle, which allows them entry into a remote filesystem.

The mount protocol is kept separate from the NFS protocol to make it easy to plug in new access checking and validation methods without changing the NFS server protocol.

Note: the protocol definition implies stateful servers because the server maintains a list of client's mount requests. The mount list information is not critical for the correct functioning of either the client or the server. It is intended for advisory use only, for example, to warn people when a server is going down.

## PROGRAMMING

**#include <rpcsvc/mount.h>**

The following XDR routines are available in **librpcsvc**:

**xdr_exportbody**
**xdr_exports**
**xdr_fhandle**
**xdr_fhstatus**
**xdr_groups**
**xdr_mountbody**
**xdr_mountlist**
**xdr_path**

## SEE ALSO

**mount**(8), **mountd**(8C), **showmount**(8)

*NFS Protocol Spec*, in *Network Programming*

**NAME**

ipalloc - determine or temporarily allocate IP address

**PROTOCOL**

**/usr/include/rpcsvc/ipalloc.x**

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

**ipalloc**( ) is the protocol for allocating the IP address that a system should use.

**PROGRAMMING**

**#include <rpcsvc/ipalloc.h>**

The following RPC calls are available in version 2 of this protocol:

**NULLPROC**

This is a standard null entry, used to ping a service to measure overhead or to discover servers.

**IP_ALLOC**

Returns an IP address corresponding to a given Ethernet address, if possible. This RPC must be called using DES authentication, from a client authorized to allocate IP addresses. A cache of allocated addresses is maintained.

The first action taken on receipt of this RPC is to verify that no existing mapping between the *etheraddr* and the *netnum* exists in the YP database. If one is found, then that is returned. Otherwise, an internal cache is checked, and if an entry is found there for the given *etheraddr* on the right network, that entry is used. If no address was found either in the YP database or in the cache, a new one may be allocated and returned, and the *ip_success* status is returned.

If an unusable entry was found in the cache, this RPC returns **ip_failure** status.

**IP_TONAME**

Used to determine whether a given IP address is known to the YP service, since YP allows a delay between the posting of an address and its availability in some locations on the network.

**IP_FREE**

This RPC is used to delete *ipaddr* entries from the cache when they are no longer needed there. It requires the same protections as the **IP_ALLOC** RPC.

**SEE ALSO**

**ipallocd**(8C), **pnpboot**(8C)

## NAME

nlm_prot – protocol between local and remote network lock managers

## PROTOCOL

**/usr/include/rpcsvc/nlm_prot.x**

## DESCRIPTION

The network lock manager protocol is used for communication between local and remote lock managers.

## PROGRAMMING

**#include <rpcsvc/nlm_prot.h>**

### XDR Routines

The following XDR routines are available in **librpcsvc**:

**xdr_nlm_testargs**
**xdr_nlm_testres**
**xdr_nlm_lockargs**
**xdr_nlm_cancargs**
**xdr_nlm_unlockargs**
**xdr_nlm_res**

## SEE ALSO

**lockd**(8C)

**NAME**

pnp - automatic network installation

**PROTOCOL**

**/usr/include/rpcsvc/pnprpc.x**

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

**pnp()** is used during unattended network installation, and routine booting, of Sun386i systems on a Sun386i network. Each network cable (subnetwork or full network) must have at least one **pnpd**(8C) server running on it to support PNP.

**PROGRAMMING**

**#include <rpcsvc/pnprpc.h>**

The following RPC calls are available in version 2 of the PNP protocol:

**NULLPROC**

Finds a PNP daemon on the local network. Used with **clntudp_broadcast()**, often to measure network overhead.

**PNP_WHOAMI**

Used early in the boot process to acquire network configuration information about a system, or to determine that a system is not known by the network.

**PNP_ACQUIRE**

Used to acquire a server willing to configure a new system after a **PNP_WHOAMI** request fails. This RPC is typically broadcast; any successful reply may be used.

**PNP_SETUP**

Requests a network configuration from a PNP daemon that has responded to a previous **PNP_ACQUIRE** RPC.

**PNP_POLL**

After a **PNP_SETUP** request, if the status is **in_progress**, the procedure is to wait 20 seconds, and issue a **PNP_POLL** request, and then check the status again. Once the status is **success**, the system will be configured for the network. Entries in the yp database may be added or old ones deleted, and file storage may be assigned, according to the architecture and boot type.

If the server misses 5 **PNP_POLL** requests, it will assume that the client system crashed and back out of the procedure. Similarly, if the client system does not receive responses from the server for **PNP_MISSEDPOLLS** consecutive requests, it should assume the server crashed and begin its PNP sequence again.

**SEE ALSO**

**pnpboot**(8C), **pnpd**(8C)

**NAME**

rnusers, rusers – return information about users on remote machines

**PROTOCOL**

**/usr/include/rpcsvc/rnusers.x**

**PROGRAMMING**

```
#include <rpcsvc/rusers.h>
rnusers(host)
        char *host
rusers(host, up)
        char *host
        struct utmpidlearr *up;
```

**rnusers()** returns the number of users logged on to *host* (–1 if it cannot determine that number). **rusers()** fills the **utmpidlearr** structure with data about *host*, and returns 0 if successful.

The following XDR routines are also available:

**xdr_utmpidle**

**xdr_utmpidlearr**

**SEE ALSO**

**rusers**(1C)

**NAME**

rquota – implement quotas on remote machines

**PROTOCOL**

**/usr/include/rpcsvc/rquota.x**

**DESCRIPTION**

The **rquota( )** protocol inquires about quotas on remote machines. It is used in conjunction with NFS, since NFS itself does not implement quotas.

**PROGRAMMING**

**#include <rpcsvc/rquota.h>**

The following XDR routines are available in **librpcsvc**:

**xdr_getquota_arg**
**xdr_getquota_rslt**
**xdr_rquota**

**SEE ALSO**

**quota**(1), **quotactl**(2)

## NAME

rstat – get performance data from remote kernel

## PROTOCOL

**/usr/include/rpcsvc/rstat.x**

## DESCRIPTION

The **rstat()** protocol is used to gather statistics from remote kernel. Statistics are available on items such as paging, swapping and cpu utilization.

## PROGRAMMING

```
#include <rpcsvc/rstat.h>

havedisk(host)
        char *host;

rstat(host, statp)
        char *host;
        struct statstime *statp;
```

**havedisk()** returns 1 if *host* has a disk, 0 if it does not, and –1 if this cannot be determined. **rstat()** fills in the **statstime** structure for *host*, and returns 0 if it was successful.

The following XDR routines are available in **librpcsvc**:

**xdr_statstime**
**xdr_statsswtch**
**xdr_stats**

## SEE ALSO

**perfmeter**(1), **rup**(1C), **rstatd**(8C)

## NAME

rwall – write to specified remote machines

## SYNOPSIS

```
#include <rpcsvc/rwall.h>

rwall(host, msg);
        char *host, *msg;
```

## DESCRIPTION

*host* prints the string *msg* to all its users. It returns 0 if successful.

## RPC INFO

**program number:**

WALLPROG

**procs:**

WALLPROC_WALL

Takes string as argument (wrapstring), returns no arguments.
Executes *wall* on remote host with string.

**versions:**

RSTATVERS_ORIG

## SEE ALSO

**rwall**(1C), **rwalld**(8C), **shutdown**(8)

**NAME**

sm_inter – status monitor protocol

**PROTOCOL**

**/usr/include/rpcsvc/sm_inter.x**

**DESCRIPTION**

The status monitor protocol is used for monitoring the status of remote hosts.

**PROGRAMMING**

**#include <rpcsvc/sm_inter.h>**

**XDR Routines**

The following XDR routines are available in **librpcsvc**:

**xdr_sm_name**
**xdr_mon**
**xdr_mon_id**
**xdr_sm_stat_res**
**xdr_sm_stat**

**SEE ALSO**

**statd**(8C)

**NAME**

spray – scatter data in order to check the network

**PROTOCOL**

**/usr/include/rpcsvc/spray.x**

**DESCRIPTION**

The spray protocol sends packets to a given machine to test the speed and reliability of it.

**PROGRAMMING**

**#include <rpcsvc/spray.h>**

The following XDR routines are available in **librpcsvc**:

**xdr_sprayarr**
**xdr_spraycumul**

**SEE ALSO**

**spray**(8C), **sprayd**(8C)

NAME

xcrypt, xdecrypt, passwd2des – hex encryption and utility routines

SYNOPSIS

```
xencrypt(data, key)
        char *data;
        char *key;

xdecrypt(data, key)
        char *data;
        char *key;

passwd2des(pass, key)
        char *pass;
        char *key;
```

DESCRIPTION

The routines **xencrypt** and **xdecrypt** take NULL-terminated hexadecimal strings as arguments, and encrypt them using the 8-byte *key* as input to the DES algorithm. The input strings must have a length that is a multiple on 16 hex digits (64 bits is the DES block size).

**passwd2des** converts a password, of arbitrary length, into an 8-byte DES key, with odd-parity set in the low bit of each byte. The high-order bit of each input byte is ignored.

These routines are used by the DES authentication subsystem for encrypting and decrypting the secret keys stored in the publickey database.

SEE ALSO

**des_crypt**(3), **publickey**(5)

**NAME**

yp – Yellow Pages protocol

**PROTOCOL**

**/usr/include/rpcsvc/yp.x**

**DESCRIPTION**

The Yellow Pages Service is used for the administration of network-wide databases. The service is composed mainly of two programs: **YPBINDPROG** for finding a YP server and **YPPROG** for accessing the YP databases.

**PROGRAMMING**

Refer to **ypclnt**(3N) for information on the programmatic interface to YP servers and databases.

**SEE ALSO**

**ypclnt**(3N), **yppasswd**(3R)

**NAME**

yppasswd – update user password in Yellow Pages

**PROTOCOL**

**/usr/include/rpcsvc/yppasswd.x**

**DESCRIPTION**

The yppasswd protocol is used to change a user's password entry in the YP password database.

**PROGRAMMING**

```
#include <rpcsvc/yppasswd.h>

yppasswd(oldpass, newpw)
        char *oldpass
        struct passwd *newpw;
```

If *oldpass* is indeed the old user password, this routine replaces the password entry with *newpw*. It returns 0 if successful.

**SEE ALSO**

**yppasswd**(1), **yppasswdd**(8C)

NAME

intro – introduction to System V functions

SYNOPSIS

**/usr/5bin/cc**

DESCRIPTION

These functions are contained in the System V library, */usr/5lib/libc.a*. They are automatically linked when you compile a C program with the C compiler in **/usr/5bin/cc**.

LIST OF SYSTEM V LIBRARY FUNCTIONS

| Name | Appears on Page | Description |
|---|---|---|
| **_tolower()** | **ctype**(3V) | character classification and conversion macros and functions |
| **_toupper()** | **ctype**(3V) | character classification and conversion macros and functions |
| **addch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **addstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **asctime()** | **ctime**(3V) | convert date and time |
| **assert()** | **assert**(3V) | verify program assertion |
| **attroff()** | **curses**(3V) | System V cursor addressing and screen display library |
| **attron()** | **curses**(3V) | System V cursor addressing and screen display library |
| **attrset()** | **curses**(3V) | System V cursor addressing and screen display library |
| **baudrate()** | **curses**(3V) | System V cursor addressing and screen display library |
| **beep()** | **curses**(3V) | System V cursor addressing and screen display library |
| **box()** | **curses**(3V) | System V cursor addressing and screen display library |
| **cbreak()** | **curses**(3V) | System V cursor addressing and screen display library |
| **clear()** | **curses**(3V) | System V cursor addressing and screen display library |
| **clearerr()** | **ferror**(3V) | stream status inquiries |
| **clearok()** | **curses**(3V) | System V cursor addressing and screen display library |
| **clrtobot()** | **curses**(3V) | System V cursor addressing and screen display library |
| **clrtoeol()** | **curses**(3V) | System V cursor addressing and screen display library |
| **ctime()** | **ctime**(3V) | convert date and time |
| **ctype()** | **ctype**(3V) | character classification and conversion macros and functions |
| **curses()** | **curses**(3V) | System V cursor addressing and screen display library |
| **curses()** | **curses**(3V) | System V cursor addressing and screen display library |
| **delay_output()** | **curses**(3V) | System V cursor addressing and screen display library |
| **delch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **deleteln()** | **curses**(3V) | System V cursor addressing and screen display library |
| **delwin()** | **curses**(3V) | System V cursor addressing and screen display library |
| **doupdate()** | **curses**(3V) | System V cursor addressing and screen display library |
| **echo()** | **curses**(3V) | System V cursor addressing and screen display library |
| **endpwent()** | **getpwent**(3V) | get password file entry |
| **endwin()** | **curses**(3V) | System V cursor addressing and screen display library |
| **erase()** | **curses**(3V) | System V cursor addressing and screen display library |
| **erasechar()** | **curses**(3V) | System V cursor addressing and screen display library |
| **fdopen()** | **fopen**(3V) | open a stream |
| **feof()** | **ferror**(3V) | stream status inquiries |
| **ferror()** | **ferror**(3V) | stream status inquiries |
| **fgetc()** | **getc**(3V) | get character or integer from stream |
| **fgetpwent()** | **getpwent**(3V) | get password file entry |
| **fileno()** | **ferror**(3V) | stream status inquiries |
| **fixterm()** | **curses**(3V) | System V cursor addressing and screen display library |
| **flash()** | **curses**(3V) | System V cursor addressing and screen display library |
| **flushinp()** | **curses**(3V) | System V cursor addressing and screen display library |
| **fopen()** | **fopen**(3V) | open a stream |

| | | |
|---|---|---|
| **fprintf()** | **printf**(3V) | formatted output conversion |
| **freopen()** | **fopen**(3V) | open a stream |
| **fscanf()** | **scanf**(3V) | formatted input conversion |
| **getc()** | **getc**(3V) | get character or integer from stream |
| **getch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **getchar()** | **getc**(3V) | get character or integer from stream |
| **getpass()** | **getpass**(3V) | read a password |
| **getpwent()** | **getpwent**(3V) | get password file entry |
| **getpwnam()** | **getpwent**(3V) | get password file entry |
| **getpwuid()** | **getpwent**(3V) | get password file entry |
| **getstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **gettmode()** | **curses**(3V) | System V cursor addressing and screen display library |
| **getw()** | **getc**(3V) | get character or integer from stream |
| **getyx()** | **curses**(3V) | System V cursor addressing and screen display library |
| **gmtime()** | **ctime**(3V) | convert date and time |
| **has_ic()** | **curses**(3V) | System V cursor addressing and screen display library |
| **has_il()** | **curses**(3V) | System V cursor addressing and screen display library |
| **idlok()** | **curses**(3V) | System V cursor addressing and screen display library |
| **inch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **initscr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **insch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **insertln()** | **curses**(3V) | System V cursor addressing and screen display library |
| **intrflush()** | **curses**(3V) | System V cursor addressing and screen display library |
| **isalnum()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isalpha()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isascii()** | **ctype**(3V) | character classification and conversion macros and funct |
| **iscntrl()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isdigit()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isgraph()** | **ctype**(3V) | character classification and conversion macros and funct |
| **islower()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isprint()** | **ctype**(3V) | character classification and conversion macros and funct |
| **ispunct()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isspace()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isupper()** | **ctype**(3V) | character classification and conversion macros and funct |
| **isxdigit()** | **ctype**(3V) | character classification and conversion macros and funct |
| **keypad()** | **curses**(3V) | System V cursor addressing and screen display library |
| **killchar()** | **curses**(3V) | System V cursor addressing and screen display library |
| **leaveok()** | **curses**(3V) | System V cursor addressing and screen display library |
| **localtime()** | **ctime**(3V) | convert date and time |
| **longname()** | **curses**(3V) | System V cursor addressing and screen display library |
| **meta()** | **curses**(3V) | System V cursor addressing and screen display library |
| **move()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvaddch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvaddstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvcur()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvdelch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvgetch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvgetstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvinch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvinsch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvprintw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvscanw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwaddch()** | **curses**(3V) | System V cursor addressing and screen display library |

| | | |
|---|---|---|
| **mvwaddstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwdelch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwgetch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwgetstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwin()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwinch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwinsch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwprintw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **mvwscanw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **newpad()** | **curses**(3V) | System V cursor addressing and screen display library |
| **newterm()** | **curses**(3V) | System V cursor addressing and screen display library |
| **newwin()** | **curses**(3V) | System V cursor addressing and screen display library |
| **nice()** | **nice**(3V) | change priority of a process |
| **nl()** | **curses**(3V) | System V cursor addressing and screen display library |
| **nocbreak()** | **curses**(3V) | System V cursor addressing and screen display library |
| **nodelay()** | **curses**(3V) | System V cursor addressing and screen display library |
| **noecho()** | **curses**(3V) | System V cursor addressing and screen display library |
| **nonl()** | **curses**(3V) | System V cursor addressing and screen display library |
| **noraw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **overlay()** | **curses**(3V) | System V cursor addressing and screen display library |
| **overwrite()** | **curses**(3V) | System V cursor addressing and screen display library |
| **pnoutrefresh()** | **curses**(3V) | System V cursor addressing and screen display library |
| **printf()** | **printf**(3V) | formatted output conversion |
| **rand()** | **rand**(3V) | simple random number generator |
| **raw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **refresh()** | **curses**(3V) | System V cursor addressing and screen display library |
| **resetterm()** | **curses**(3V) | System V cursor addressing and screen display library |
| **resetty()** | **curses**(3V) | System V cursor addressing and screen display library |
| **saveterm()** | **curses**(3V) | System V cursor addressing and screen display library |
| **savetty()** | **curses**(3V) | System V cursor addressing and screen display library |
| **scanf()** | **scanf**(3V) | formatted input conversion |
| **scanw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **scroll()** | **curses**(3V) | System V cursor addressing and screen display library |
| **scrollok()** | **curses**(3V) | System V cursor addressing and screen display library |
| **set_term()** | **curses**(3V) | System V cursor addressing and screen display library |
| **setbuf()** | **setbuf**(3V) | assign buffering to a stream |
| **setbuffer()** | **setbuf**(3V) | assign buffering to a stream |
| **setgid()** | **setuid**(3V) | set user and group IDs |
| **setlinebuf()** | **setbuf**(3V) | assign buffering to a stream |
| **setpwent()** | **getpwent**(3V) | get password file entry |
| **setpwfile()** | **getpwent**(3V) | get password file entry |
| **setscrreg()** | **curses**(3V) | System V cursor addressing and screen display library |
| **setterm()** | **curses**(3V) | System V cursor addressing and screen display library |
| **setuid()** | **setuid**(3V) | set user and group IDs |
| **setupterm()** | **curses**(3V) | System V cursor addressing and screen display library |
| **setvbuf()** | **setbuf**(3V) | assign buffering to a stream |
| **signal()** | **signal**(3V) | simplified software signal facilities |
| **sleep()** | **sleep**(3V) | suspend execution for interval |
| **sprintf()** | **printf**(3V) | formatted output conversion |
| **srand()** | **rand**(3V) | simple random number generator |
| **sscanf()** | **scanf**(3V) | formatted input conversion |
| **standend()** | **curses**(3V) | System V cursor addressing and screen display library |
| **standout()** | **curses**(3V) | System V cursor addressing and screen display library |

| | | |
|---|---|---|
| **subwin()** | **curses**(3V) | System V cursor addressing and screen display library |
| **timegm()** | **ctime**(3V) | convert date and time |
| **timelocal()** | **ctime**(3V) | convert date and time |
| **times()** | **times**(3V) | get process and child process times |
| **toascii()** | **ctype**(3V) | character classification and conversion macros and functions |
| **tolower()** | **ctype**(3V) | character classification and conversion macros and functions |
| **touchwin()** | **curses**(3V) | System V cursor addressing and screen display library |
| **toupper()** | **ctype**(3V) | character classification and conversion macros and functions |
| **traceoff()** | **curses**(3V) | System V cursor addressing and screen display library |
| **traceon()** | **curses**(3V) | System V cursor addressing and screen display library |
| **ttyslot()** | **ttyslot**(3V) | find the slot in the utmp file of the current process |
| **typeahead()** | **curses**(3V) | System V cursor addressing and screen display library |
| **tzset()** | **ctime**(3V) | convert date and time |
| **tzsetwall()** | **ctime**(3V) | convert date and time |
| **unctrl()** | **curses**(3V) | System V cursor addressing and screen display library |
| **waddch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **waddstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wattroff()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wattron()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wattrset()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wclear()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wclrtobot()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wclrtoeol()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wdelch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wdeleteln()** | **curses**(3V) | System V cursor addressing and screen display library |
| **werase()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wgetch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wgetstr()** | **curses**(3V) | System V cursor addressing and screen display library |
| **winch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **winsch()** | **curses**(3V) | System V cursor addressing and screen display library |
| **winsertln()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wmove()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wnoutrefresh()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wprintw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wrefresh()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wscanw()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wsetscrreg()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wstandend()** | **curses**(3V) | System V cursor addressing and screen display library |
| **wstandout()** | **curses**(3V) | System V cursor addressing and screen display library |

## NAME

assert – verify program assertion

## SYNOPSIS

**#include <assert.h>**

**assert(expression)**
**int expression;**

## DESCRIPTION

**assert( )** is a macro that indicates *expression* is expected to be true at this point in the program. When it is executed, if *expression* is false (zero), **assert( )** prints

'**Assertion failed:** *expression*, **file** *xyz*, **line** *nnn*'

on the standard error output and aborts. In the error message, *xyz* is the name of the source file and *nnn* the source line number of the **assert( )** statement.

Compiling with the **cc**(1V) option **–DNDEBUG**, or with the preprocessor control statement '**#define NDEBUG**' ahead of the '**#include <assert.h>**' statement, will stop assertions from being compiled into the program.

## SEE ALSO

**cc**(1V), **abort**(3)

NAME

ctime, localtime, gmtime, asctime, timelocal, timegm, tzset, tzsetwall – convert date and time

SYNOPSIS

```
#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *ctime(clock)
long *clock;

time_t timelocal(tm)
struct tm *tm;

time_t timegm(tm)
struct tm *tm;

void tzset()

void tzsetwall()

extern long timezone;

extern int daylight;

extern char *tzname[2];
```

DESCRIPTION

**localtime**() and **gmtime**() return pointers to structures containing the time, broken down into various components of that time represented in a particular time zone. **localtime**() breaks down a time specified by the **clock**() argument, correcting for the time zone and any time zone adjustments (such as Daylight Savings Time). Before doing so, **localtime**() calls **tzset**() (if **tzset**() has not been called in the current process). **gmtime**() breaks down a time specified by the **clock**() argument into GMT, which is the time the system uses.

**asctime**() converts a time value contained in a "tm" structure to a 26-character string of the form:

**Sun Sep 16 01:03:52 1973\n\0**

Each field has a constant width. **asctime**() returns a pointer to the string.

**ctime**() converts a long integer, pointed to by *clock*, to a 26-character string of the form produced by **asctime**(). It first breaks down **clock**() to a **struct tm** by calling **localtime**, and then calls **asctime**() to convert that **struct tm** to a string.

**timelocal**() and **timegm**() convert the time specified by the **tm** argument to a time value that represents that time expressed as the number of seconds since Jan. 1, 1970, 00:00, Greenwich Mean Time. **timelocal**() converts a **struct tm** that represents local time, correcting for the time zone and any time zone adjustments (such as Daylight Savings Time). Before doing so, **timelocal**() calls **tzset**() (if **tzset**() has not been called in the current process). **timegm**() converts a **struct tm** that represents GMT.

**tzset**() uses the value of the environment variable TZ to set time conversion information used by *localtime*. If TZ is absent from the environment, the best available approximation to local wall clock time is used by *localtime*. If TZ appears in the environment but its value is a NULL string, Greenwich Mean Time is used; if TZ appears and begins with a slash, it is used as the absolute pathname of the *tzfile-format* (see **tzfile**(5)) file from which to read the time conversion information; if TZ appears and begins with a character other than a slash, it is used as a pathname relative to a system time conversion information directory.

**tzsetwall()** sets things up so that **localtime()** returns the best available approximation of local wall clock time.

Declarations of all the functions and externals, and the "tm" structure, are in the **<time.h>** header file. The structure (of type) **struct tm** includes the following fields:

```
int tm_sec;     /* seconds (0 - 59) */
int tm_min;     /* minutes (0 - 59) */
int tm_hour;    /* hours (0 - 23) */
int tm_mday;    /* day of month (1 - 31) */
int tm_mon;     /* month of year (0 - 11) */
int tm_year;    /* year - 1900 */
int tm_wday;    /* day of week (Sunday = 0) */
int tm_yday;    /* day of year (0 - 365) */
int tm_isdst;   /* 1 if DST in effect */
char *tm_zone;  /* abbreviation of timezone name */
long tm_gmtoff; /* offset from GMT in seconds */
```

**tm_isdst** is non-zero if Daylight Savings Time is in effect. **tm_zone** points to a string that is the name used for the local time zone at the time being converted. **tm_gmtoff** is the offset (in seconds) of the time represented from GMT, with positive values indicating East of Greenwich.

The external **long** variable *timezone* contains the difference, in seconds, between GMT and local standard time (in PST, *timezone* is 8*60*60). If this difference is not a constant, *timezone* will contain the value of the offset on January 1, 1970 at 00:00 GMT. Since this is not necessarily the same as the value at some particular time, the time in question should be converted to a "struct tm" using **localtime** (see **ctime**(3)) and the **tm_gmtoff** field of that structure should be used. The external variable *daylight* is non-zero if and only if Daylight Savings Time would be in effect within the current time zone at some time; it does not indicate whether Daylight Savings Time is currently in effect.

The external variable *tzname* is an array of two **char** * pointers. The first pointer points to a character string that is the name of the current time zone when Daylight Savings Time is not in effect; the second one, if Daylight Savings Time conversion should be applied, points to a character string that is the name of the current time zone when Daylight Savings Time is in effect. These strings are updated by **localtime()** whenever a time is converted. If Daylight Savings Time is in effect at the time being converted, the second pointer is set to point to the name of the current time zone at that time, otherwise the first pointer is so set.

*timezone*, *daylight*, and *tzname* are retained for compatibility with existing programs.

FILES

**/usr/share/lib/zoneinfo** standard time conversion information directory
**/usr/share/lib/zoneinfo/localtime**
local time zone file

SEE ALSO

**gettimeofday**(2), **ctime**(3), **getenv**(3), **time**(3C), **environ**(5V), **tzfile**(5)

BUGS

The return values point to static data, whose contents are overwritten by each call. The **tm_zone** field of a returned **struct tm** points to a static array of characters, which will also be overwritten at the next call (and by calls to **tzset()** or *tzsetwall*).

NAME

ctype, isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii, isgraph, toupper, tolower, toascii, _toupper, _tolower – character classification and conversion macros and functions

SYNOPSIS

**#include <ctype.h>**

**isalpha(c)**

...

CHARACTER CLASSIFICATION MACROS

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *isascii* is defined on all integer values; the rest are defined only where **isascii**(*c*) is true and on the single non-ASCII value EOF (see **stdio**(3V)).

| | |
|---|---|
| **isalpha**(*c*) | *c* is a letter |
| **isupper**(*c*) | *c* is an upper case letter |
| **islower**(*c*) | *c* is a lower case letter |
| **isdigit**(*c*) | *c* is a digit [0-9]. |
| **isxdigit**(*c*) | *c* is a hexadecimal digit [0-9], [A-F], or [a-f]. |
| **isalnum**(*c*) | *c* is an alphanumeric character, that is, *c* is a letter or a digit |
| **isspace**(*c*) | *c* is a space, tab, carriage return, newline, vertical tab, or formfeed |
| **ispunct**(*c*) | *c* is a punctuation character (neither control nor alphanumeric) |
| **isprint**(*c*) | *c* is a printing character, code 040(8) (space) through 0176 (tilde) |
| **iscntrl**(*c*) | *c* is a delete character (0177) or ordinary control character (less than 040). |
| **isascii**(*c*) | *c* is an ASCII character, code less than 0200 |
| **isgraph**(*c*) | *c* is a visible graphic character, code 041 (exclamation mark) through 0176 (tilde). |

CHARACTER CONVERSION MACROS AND FUNCTIONS

*toupper* and *tolower* are functions, rather than macros, and work correctly on all characters. The macros *_toupper* and *_tolower* are faster than the equivalent functions (*toupper* and *tolower*) but only work properly on a restricted range of characters.

These functions perform simple conversions on single characters.

| | |
|---|---|
| **toupper**(*c*) | converts *c* to its upper-case equivalent. If *c* is not a lower-case letter, it is returned unchanged. |
| **tolower**(*c*) | converts *c* to its lower-case equivalent. If *c* is not an upper-case letter, it is returned unchanged. |
| **toascii**(*c*) | masks *c* with the correct value so that *c* is guaranteed to be an ASCII character in the range 0 thru 0x7f. |

These macros perform simple conversions on single characters.

| | |
|---|---|
| **_toupper**(*c*) | converts *c* to its upper-case equivalent. Note that this *only* works where *c* is known to be a lower-case character to start with (presumably checked using *islower*). |
| **_tolower**(*c*) | converts *c* to its lower-case equivalent. Note: this *only* works where *c* is known to be a upper-case character to start with (presumably checked using *isupper*). |

**DIAGNOSTICS**

If the argument to any of these macros is not in the domain of the function, the result is undefined.

**SEE ALSO**

**ctype**(3), **stdio**(3V), **ascii**(7)

NAME

curses – System V terminal screen handling and optimization package

SYNOPSIS

The **curses** manual page is organized as follows:

In SYNOPSIS

- compiling information
- summary of parameters used by **curses** routines
- alphabetical list of **curses** routines, showing their parameters

In DESCRIPTION:

- An overview of how **curses** routines should be used

In ROUTINES, descriptions of **curses** routines are grouped under the appropriate topics:

- Overall Screen Manipulation
- Window and Pad Manipulation
- Output
- Input
- Output Options Setting
- Input Options Setting
- Environment Queries
- Soft Labels
- Low-level Curses Access
- Terminfo-Level Manipulations
- Termcap Emulation
- Miscellaneous
- Use of **curscr**

Then come sections on:

- ATTRIBUTES
- FUNCTION CALLS
- LINE GRAPHICS

**/usr/5bin/cc** [ *flag* ...] *file* ... **–lcurses** [ *library* ...]

**#include <curses.h>** (automatically includes **<stdio.h>**, **<termio.h>**, and **<unctrl.h>**).

The parameters in the following list are not global variables; this is a summary of the parameters used by the **curses** library routines. All routines return the **int** values **ERR** or **OK** unless otherwise noted. Routines that return pointers always return NULL on error. (**ERR**, **OK**, and **NULL** are all defined in **<curses.h>**.) Routines that return integers are not listed in the parameter list below.

**bool bf**

**char **area,*boolnames[ ], *boolcodes[ ], *boolfnames[ ], *bp**
**char *cap, *capname, codename[2], erasechar, *filename, *fmt**
**char *keyname, killchar, *label, *longname**
**char *name, *numnames[ ], *numcodes[ ], *numfnames[ ]**
**char *slk_label, *str, *strnames[ ], *strcodes[ ], *strfnames[ ]**
**char *term, *tgetstr, *tigetstr, *tgoto, *tparm, *type**

**chtype attrs, ch, horch, vertch**

**FILE *infd, *outfd**

**int begin_x, begin_y, begline, bot, c, col, count**
**int dmaxcol, dmaxrow, dmincol, dminrow, *errret, fildes**
**int (*init( )), labfmt, labnum, line**
**int ms, ncols, new, newcol, newrow, nlines, numlines**

**int oldcol, oldrow, overlay**
**int p1, p2, p9, pmincol, pminrow, (*putc()), row**
**int smaxcol, smaxrow, smincol, sminrow, start**
**int tenths, top, visibility, x, y**

**SCREEN *new, *newterm, *set_term**

**TERMINAL *cur_term, *nterm, *oterm**

**va_list varglist**

**WINDOW *curscr, *dstwin, *initscr, *newpad, *newwin, *orig**

**WINDOW *pad, *srcwin, *stdscr, *subpad, *subwin, *win**

**addch** (*ch*)
**addstr** (*str*)
**attroff** (*attrs*)
**attron** (*attrs*)
**attrset** (*attrs*)
**baudrate()**
**beep()**
**box** (*win, vertch, horch*)
**cbreak()**
**clear()**
**clearok** (*win, bf*)
**clrtobot()**
**clrtoeol()**
**copywin** (*srcwin, dstwin, sminrow, smincol, dminrow, dmincol, dmaxrow, dmaxcol, overlay*)
**curs_set** (*visibility*)
**def_prog_mode()**
**def_shell_mode()**
**del_curterm** (*oterm*)
**delay_output** (*ms*)
**delch()**
**deleteln()**
**delwin** (*win*)
**doupdate()**
**draino** (*ms*)
**echo()**
**echochar** (*ch*)
**endwin()**
**erase()**
**erasechar()**
**filter()**
**flash()**
**flushinp()**
**garbagedlines** (*win, begline, numlines*)
**getbegyx** (*win, y, x*)
**getch()**
**getmaxyx** (*win, y, x*)
**getstr** (*str*)
**getsyx** (*y, x*)
**getyx** (*win, y, x*)
**halfdelay** (*tenths*)
**has_ic()**
**has_il()**
**idlok** (*win, bf*)

**inch()**
**initscr()**
**insch** (*ch*)
**insertln()**
**intrflush** (*win, bf*)
**isendwin()**
**keyname** (*c*)
**keypad** (*win, bf*)
**killchar()**
**leaveok** (*win, bf*)
**longname()**
**meta** (*win, bf*)
**move** (*y, x*)
**mvaddch** (*y, x, ch*)
**mvaddstr** (*y, x, str*)
**mvcur** (*oldrow, oldcol, newrow, newcol*)
**mvdelch** (*y, x*)
**mvgetch** (*y, x*)
**mvgetstr** (*y, x, str*)
**mvinch** (*y, x*)
**mvinsch** (*y, x, ch*)
**mvprintw** (*y, x, fmt* [, *arg* ...])
**mvscanw** (*y, x, fmt* [, *arg* ...])
**mvwaddch** (*win, y, x, ch*)
**mvwaddstr** (*win, y, x, str*)
**mvwdelch** (*win, y, x*)
**mvwgetch** (*win, y, x*)
**mvwgetstr** (*win, y, x, str*)
**mvwin** (*win, y, x*)
**mvwinch** (*win, y, x*)
**mvwinsch** (*win, y, x, ch*)
**mvwprintw** (*win, y, x, fmt* [, *arg* ...])
**mvwscanw** (*win, y, x, fmt* [, *arg* ...])
**napms** (*ms*)
**newpad** (*nlines, ncols*)
**newterm** (*type, outfd, infd*)
**newwin** (*nlines, ncols, begin_y, begin_x*)
**nl()**
**nocbreak()**
**nodelay** (*win, bf*)
**noecho()**
**nonl()**
**noraw()**
**notimeout** (*win, bf*)
**overlay** (*srcwin, dstwin*)
**overwrite** (*srcwin, dstwin*)
**pechochar** (*pad, ch*)
**pnoutrefresh** (*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol*)
**prefresh** (*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol*)
**printw** (*fmt* [, *arg* ...])
**putp** (*str*)
**raw()**
**refresh()**

**reset_prog_mode( )**
**reset_shell_mode( )**
**resetty( )**
**restartterm** (*term, fildes, errret*)
**ripoffline** (*line, init*)
**savetty( )**
**scanw** (*fmt* [, *arg* . . .])
**scr_dump** (*filename*)
**scr_init** (*filename*)
**scr_restore** (*filename*)
**scroll** (*win*)
**scrollok** (*win, bf*)
**set_curterm** (*nterm*)
**set_term** (*new*)
**setscrreg** (*top, bot*)
**setsyx** (*y, x*)
**setupterm** (*term, fildes, errret*)
**slk_clear( )**
**slk_init** (*fmt*)
**slk_label** (*labnum*)
**slk_noutrefresh( )**
**slk_refresh( )**
**slk_restore( )**
**slk_set** (*labnum, label, fmt*)
**slk_touch( )**
**standend( )**
**standout( )**
**subpad** (*orig, nlines, ncols, begin_y, begin_x*)
**subwin** (*orig, nlines, ncols, begin_y, begin_x*)
**tgetent** (*bp, name*)
**tgetflag** (*codename*)
**tgetnum** (*codename*)
**tgetstr** (*codename, area*)
**tgoto** (*cap, col, row*)
**tigetflag** (*capname*)
**tigetnum** (*capname*)
**tigetstr** (*capname*)
**touchline** (*win, start, count*)
**touchwin** (*win*)
**tparm** (*str, p1, p2, . . ., p9*)
**tputs** (*str, count, putc*)
**traceoff( )**
**traceon( )**
**typeahead** (*fildes*)
**unctrl** (*c*)
**ungetch** (*c*)
**vidattr** (*attrs*)
**vidputs** (*attrs, putc*)
**vwprintw** (*win, fmt, varglist*)
**vwscanw** (*win, fmt, varglist*)
**waddch** (*win, ch*)
**waddstr** (*win, str*)
**wattroff** (*win, attrs*)

**wattron** (*win, attrs*)
**wattrset** (*win, attrs*)
**wclear** (*win*)
**wclrtobot** (*win*)
**wclrtoeol** (*win*)
**wdelch** (*win*)
**wdeleteln** (*win*)
**wechochar** (*win, ch*)
**werase** (*win*)
**wgetch** (*win*)
**wgetstr** (*win, str*)
**winch** (*win*)
**winsch** (*win, ch*)
**winsertln** (*win*)
**wmove** (*win, y, x*)
**wnoutrefresh** (*win*)
**wprintw** (*win, fmt* [, *arg* ...])
**wrefresh** (*win*)
**wscanw** (*win, fmt* [, *arg* ...])
**wsetscrreg** (*win, top, bot*)
**wstandend** (*win*)
**wstandout** (*win*)

DESCRIPTION

The **curses** routines give the user a terminal-independent method of updating screens with reasonable optimization.

In order to initialize the routines, the routine **initscr()** or **newterm()** must be called before any of the other routines that deal with windows and screens are used. (Three exceptions are noted where they apply.) The routine **endwin()** must be called before exiting. To get character-at-a-time input without echoing, (most interactive, screen oriented programs want this) after calling **initscr()** you should call '**cbreak (); noecho ();**' Most programs would additionally call '**nonl (); intrflush(stdscr, FALSE); keypad(stdscr, TRUE);**'.

Before a **curses** program is run, a terminal's TAB stops should be set and its initialization strings, if defined, must be output. This can be done by executing the **tset** command in your **.profile** or **.login** file. For further details, see **tset**(1) and the **Tabs and Initialization** subsection of **terminfo**(5V).

The **curses** library contains routines that manipulate data structures called *windows* that can be thought of as two-dimensional arrays of characters representing all or part of a terminal screen. A default window called **stdscr** is supplied, which is the size of the terminal screen. Others may be created with **newwin()**. Windows are referred to by variables declared as **WINDOW ***; the type **WINDOW** is defined in **<curses.h>** to be a C structure. These data structures are manipulated with routines described below, among which the most basic are **move()** and **addch()**. (More general versions of these routines are included with names beginning with **w**, allowing you to specify a window. The routines not beginning with **w** usually affect **stdscr**.) Then **refresh()** is called, telling the routines to make the user's terminal screen look like **stdscr**. The characters in a window are actually of type **chtype**, so that other information about the character may also be stored with each character.

Special windows called *pads* may also be manipulated. These are windows that are not constrained to the size of the screen and whose contents need not be displayed completely. See the description of **newpad()** under **Window and Pad Manipulation** for more information.

In addition to drawing characters on the screen, video attributes may be included that cause the characters to show up in modes such as underlined or in reverse video on terminals that support such display enhancements. Line drawing characters may be specified to be output. On input, **curses** is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in **<curses.h>**, such as **A_REVERSE**, **ACS_HLINE**, and

KEY_LEFT.

**curses** also defines the WINDOW * variable, **curscr**, which is used only for certain low-level operations like clearing and redrawing a garbaged screen. **curscr** can be used in only a few routines. If the window argument to **clearok( )** is **curscr**, the next call to **wrefresh( )** with any window will clear and repaint the screen from scratch. If the window argument to **wrefresh( )** is **curscr**, the screen in immediately cleared and repainted from scratch. This is how most programs would implement a "repaint-screen" function. More information on using **curscr** is provided where its use is appropriate.

The environment variables **LINES** and **COLUMNS** may be set to override **curses**'s idea of how large a screen is.

If the environment variable **TERMINFO** is defined, any program using **curses** will check for a local terminal definition before checking in the standard place. For example, if the environment variable **TERM** is set to **sun**, then the compiled terminal definition is found in **/usr/share/lib/terminfo/s/sun**. (The **s** is copied from the first letter of **sun** to avoid creation of huge directories.) However, if **TERMINFO** is set to **$HOME/myterms**, **curses** will first check **$HOME/myterms/s/sun**, and, if that fails, will then check **/usr/share/lib/terminfo/s/sun**. This is useful for developing experimental definitions or when write permission on **/usr/share/lib/terminfo** is not available.

The integer variables **LINES** and **COLS** are defined in **<curses.h>**, and will be filled in by **initscr( )** with the size of the screen. (For more information, see the subsection **Terminfo-Level Manipulations**.) The constants **TRUE** and **FALSE** have the values **1** and **0**, respectively. The constants **ERR** and **OK** are returned by routines to indicate whether the routine successfully completed. These constants are also defined in **<curses.h>**.

## ROUTINES

Many of the following routines have two or more versions. The routines prefixed with **w** require a *window* argument. The routines prefixed with **p** require a *pad* argument. Those without a prefix generally use **stdscr**.

The routines prefixed with **mv** require *y* and *x* coordinates to move to before performing the appropriate action. The **mv** routines imply a call to **move( )** before the call to the other routine. The window argument is always specified before the coordinates. *y* always refers to the row (of the window), and *x* always refers to the column. The upper left corner is always **(0,0)**, not **(1,1)**. The routines prefixed with **mvw** take both a *window* argument and *y* and *x* coordinates.

In each case, *win* is the window affected and *pad* is the pad affected. (**win** and *pad* are always of type **WINDOW ***.) Option-setting routines require a boolean flag *bf* with the value **TRUE** or **FALSE**. (*bf* is always of type **bool**.) The types **WINDOW**, **bool**, and **chtype** are defined in **<curses.h>**. See the SYNOPSIS for a summary of what types all variables are.

All routines return either the integer **ERR** or the integer **OK**, unless otherwise noted. Routines that return pointers always return NULL on error.

### Overall Screen Manipulation

**WINDOW *initscr( )** The first routine called should almost always be **initscr( )**. (The exceptions are **slk_init( )**, **filter( )**, and **ripoffline( )**.) This will determine the terminal type and initialize all **curses** data structures. **initscr( )** also arranges that the first call to **refresh( )** will clear the screen. If errors occur, **initscr( )** will write an appropriate error message to standard error and exit; otherwise, a pointer to **stdscr** is returned. If the program wants an indication of error conditions, **newterm( )** should be used instead of **initscr( )**. **initscr( )** should only be called once per application.

**endwin( )** A program should always call **endwin( )** before exiting or escaping from **curses** mode temporarily, to do a shell escape or **system**(3) call, for example. This routine will restore **termio**(4) modes, move the cursor to the lower left corner of the screen and reset the terminal into the proper non-visual mode. To resume after a temporary escape, call **wrefresh( )** or **doupdate( )**.

**isendwin()** Returns TRUE if **endwin()** has been called without any subsequent calls to **wrefresh()**.

**SCREEN *newterm(***type, outfd, infd***)**

A program that outputs to more than one terminal must use **newterm()** for each terminal instead of **initscr()**. A program that wants an indication of error conditions, so that it may continue to run in a line-oriented mode if the terminal cannot support a screen-oriented program, must also use this routine. **newterm()** should be called once for each terminal. It returns a variable of type SCREEN* that should be saved as a reference to that terminal. The arguments are the *type* of the terminal to be used in place of the environment variable TERM; *outfd*, a **stdio**(3V) file pointer for output to the terminal; and *infd*, another file pointer for input from the terminal. When it is done running, the program must also call **endwin()** for each terminal being used. If **newterm()** is called more than once for the same terminal, the first terminal referred to must be the last one for which **endwin()** is called.

**SCREEN *set_term (***new***)**

This routine is used to switch between different terminals. The screen reference *new* becomes the new current terminal. A pointer to the screen of the previous terminal is returned by the routine. This is the only routine that manipulates SCREEN pointers; all other routines affect only the current terminal.

**Window and Pad Manipulation**

**refresh()**

**wrefresh (***win***)** These routines (or **prefresh()**, **pnoutrefresh()**, **wnoutrefresh()**, or **doupdate()**) must be called to write output to the terminal, as most other routines merely manipulate data structures. **wrefresh()** copies the named window to the physical terminal screen, taking into account what is already there in order to minimize the amount of information that's sent to the terminal (called optimization). **refresh()** does the same thing, except it uses **stdscr** as a default window. Unless **leaveok()** has been enabled, the physical cursor of the terminal is left at the location of the window's cursor. The number of characters output to the terminal is returned.

Note: **refresh()** is a macro.

**wnoutrefresh (***win***)**

**doupdate()** These two routines allow multiple updates to the physical terminal screen with more efficiency than **wrefresh()** alone. How this is accomplished is described in the next paragraph.

**curses** keeps two data structures representing the terminal screen: a *physical* terminal screen, describing what is actually on the screen, and a *virtual* terminal screen, describing what the programmer wants to have on the screen. **wrefresh()** works by first calling **wnoutrefresh()**, which copies the named window to the virtual screen, and then by calling **doupdate()**, which compares the virtual screen to the physical screen and does the actual update. If the programmer wishes to output several windows at once, a series of calls to **wrefresh()** will result in alternating calls to **wnoutrefresh()** and **doupdate()**, causing several bursts of output to the screen. By first calling **wnoutrefresh()** for each window, it is then possible to call **doupdate()** once, resulting in only one burst of output, with probably fewer total characters transmitted and certainly less processor time used.

**WINDOW *newwin (***nlines, ncols, begin_y, begin_x***)**

Create and return a pointer to a new window with the given number of lines (or rows), *nlines*, and columns, *ncols*. The upper left corner of the window is at line *begin_y*, column *begin_x*. If either *nlines* or *ncols* is **0**, they will be set to the

value of **lines**–*begin_y* and **cols**–*begin_x*. A new full-screen window is created by calling **newwin(0,0,0,0)**.

**mvwin** (*win, y, x*) Move the window so that the upper left corner will be at position (*y, x*). If the move would cause the window to be off the screen, it is an error and the window is not moved.

**WINDOW *subwin** (*orig, nlines, ncols, begin_y, begin_x*)
Create and return a pointer to a new window with the given number of lines (or rows), *nlines*, and columns, *ncols*. The window is at position ( *begin_y, begin_x*) on the screen. (This position is relative to the screen, and not to the window *orig*.) The window is made in the middle of the window *orig*, so that changes made to one window will affect both windows. When using this routine, often it will be necessary to call **touchwin( )** or **touchline( )** on *orig* before calling **wrefresh**.

**delwin** (*win*) Delete the named window, freeing up all memory associated with it. In the case of overlapping windows, subwindows should be deleted before the main window.

**WINDOW *newpad** (*nlines, ncols*)
Create and return a pointer to a new pad data structure with the given number of lines (or rows), *nlines*, and columns, *ncols*. A pad is a window that is not restricted by the screen size and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (for example, from scrolling or echoing of input) do not occur. It is not legal to call **wrefresh( )** with a pad as an argument; the routines **prefresh( )** or **pnoutrefresh( )** should be called instead. Note: these routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for display.

**WINDOW *subpad** (*orig, nlines, ncols, begin_y, begin_x*)
Create and return a pointer to a subwindow within a pad with the given number of lines (or rows), *nlines*, and columns, *ncols*. Unlike **subwin( )**, which uses screen coordinates, the window is at position (*begin_y, begin_x*) on the pad. The window is made in the middle of the window *orig*, so that changes made to one window will affect both windows. When using this routine, often it will be necessary to call **touchwin( )** or **touchline( )** on *orig* before calling **prefresh( )**.

**prefresh** (*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol*)
**pnoutrefresh** (*pad, pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol*)
These routines are analogous to **wrefresh( )** and **wnoutrefresh( )** except that pads, instead of windows, are involved. The additional parameters are needed to indicate what part of the pad and screen are involved. *pminrow* and *pmincol* specify the upper left corner, in the pad, of the rectangle to be displayed. *sminrow*, *smincol*, *smaxrow*, and *smaxcol* specify the edges, on the screen, of the rectangle to be displayed in. The lower right corner in the pad of the rectangle to be displayed is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures. Negative values of *pminrow*, *pmincol*, *sminrow*, or *smincol* are treated as if they were zero.

**Output**

These routines are used to "draw" text on windows.

**addch** (*ch*)
**waddch** (*win, ch*)
**mvaddch** (*y, x, ch*)
**mvwaddch** (*win, y, x, ch*)

The character *ch* is put into the window at the current cursor position of the window and the position of the window cursor is advanced. Its function is similar to that of **putchar**() (see **putc**(3S)). At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if **scrollok**() is enabled, the scrolling region will be scrolled up one line.
If *ch* is a TAB, NEWLINE, or backspace, the cursor will be moved appropriately within the window. A NEWLINE also does a **clrtoeol**() before moving. TAB characters are considered to be at every eighth column. If *ch* is another control character, it will be drawn in the CTRL-X notation. (Calling **winch**() after adding a control character will not return the control character, but instead will return the representation of the control character.)
Video attributes can be combined with a character by or-ing them into the parameter. This will result in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another using **inch**() and **addch**().) See **standout**(), below.
Note: *ch* is actually of type **chtype**, not a character.
Note: **addch**(), **mvaddch**(), and **mvwaddch**() are macros.

**echochar** (*ch*)
**wechochar** (*win, ch*)
**pechochar** (*pad, ch*)

These routines are functionally equivalent to a call to **addch** (*ch*) followed by a call to **refresh**(), a call to **waddch** (*win, ch*) followed by a call to **wrefresh** (*win*), or a call to **waddch** (*pad, ch*) followed by a call to **prefresh** (*pad*). The knowledge that only a single character is being output is taken into consideration and, for non-control characters, a considerable performance gain can be seen by using these routines instead of their equivalents. In the case of **pechochar**(), the last location of the pad on the screen is reused for the arguments to **prefresh**().
Note: *ch* is actually of type **chtype**, not a character.
Note: **echochar**() is a macro.

**addstr** (*str*)
**waddstr** (*win, str*)
**mvwaddstr** (*win, y, x, str*)
**mvaddstr** (*y, x, str*)

These routines write all the characters of the null-terminated character string *str* on the given window. This is equivalent to calling **waddch**() once for each character in the string.
Note: **addstr**(), **mvaddstr**(), and **mvwaddstr**() are macros.

**attroff** (*attrs*)
**wattroff** (*win, attrs*)
**attron** (*attrs*)
**wattron** (*win, attrs*)
**attrset** (*attrs*)
**wattrset** (*win, attrs*)
**standend**()
**wstandend** (*win*)
**standout**()
**wstandout** (*win*)

These routines manipulate the current attributes of the named window. These

attributes can be any combination of **A_STANDOUT**, **A_REVERSE**, **A_BOLD**, **A_DIM**, **A_BLINK**, **A_UNDERLINE**, and **A_ALTCHARSET**. These constants are defined in **<curses.h>** and can be combined with the C logical OR ( | ) operator.
The current attributes of a window are applied to all characters that are written into the window with **waddch( )**. Attributes are a property of the character, and move with the character through any scrolling and insert/delete line/character operations. To the extent possible on the particular terminal, they will be displayed as the graphic rendition of the characters put on the screen.
**attrset** (*attrs*) sets the current attributes of the given window to *attrs*. **attroff** (*attrs*) turns off the named attributes without turning on or off any other attributes. **attron** (*attrs*) turns on the named attributes without affecting any others. **standout( )** is the same as **attron(A_STANDOUT)**. **standend( )** is the same as **attrset(0)**, that is, it turns off all attributes.
Note: *attrs* is actually of type **chtype**, not a character.
Note: **attroff( )**, **attron( )**, **attrset( )**, **standend( )**, and **standout( )** are macros.

**beep( )**
**flash( )**

These routines are used to signal the terminal user. **beep( )** will sound the audible alarm on the terminal, if possible, and if not, will flash the screen (visible bell), if that is possible. **flash( )** will flash the screen, and if that is not possible, will sound the audible signal. If neither signal is possible, nothing will happen. Nearly all terminals have an audible signal (bell or beep) but only some can flash the screen.

**box** (*win, vertch, horch*)

A box is drawn around the edge of the window, *win*. *vertch* and *horch* are the characters the box is to be drawn with. If *vertch* and *horch* are **0**, then appropriate default characters, **ACS_VLINE** and **ACS_HLINE**, will be used.

Note: *vertch* and *horch* are actually of type **chtype**, not characters.

**erase( )**
**werase** (*win*)

These routines copy blanks to every position in the window.
Note: **erase( )** is a macro.

**clear( )**
**wclear** (*win*)

These routines are like **erase( )** and **werase( )**, but they also call **clearok( )**, arranging that the screen will be cleared completely on the next call to **wrefresh( )** for that window, and repainted from scratch.
Note: **clear( )** is a macro.

**clrtobot( )**
**wclrtobot** (*win*)

All lines below the cursor in this window are erased. Also, the current line to the right of the cursor, inclusive, is erased.
Note: **clrtobot( )** is a macro.

**clrtoeol( )**
**wclrtoeol** (*win*)

The current line to the right of the cursor, inclusive, is erased.
Note: **clrtoeol( )** is a macro.

**delay_output** (*ms*)

Insert a *ms* millisecond pause in the output. It is not recommended that this routine be used extensively, because padding characters are used rather than a processor pause.

**delch( )**
**wdelch** (*win*)
**mvdelch** (*y, x*)
**mvwdelch** (*win, y, x*)

The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position and the last character on the line is filled with a blank. The cursor position does not change (after moving

to (*y*, *x*), if specified). (This does not imply use of the hardware "delete-character" feature.)
Note: **delch( )**, **mvdelch( )**, and **mvwdelch( )** are macros.

**deleteln( )**
**wdeleteln** (*win*)
The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. The cursor position does not change. (This does not imply use of the hardware "delete-line" feature.)
Note: **deleteln( )** is a macro.

**getyx** (*win*, *y*, *x*)
The cursor position of the window is placed in the two integer variables *y* and *x*. This is implemented as a macro, so no '**&**' is necessary before the variables.

**getbegyx** (*win*, *y*, *x*)
**getmaxyx** (*win*, *y*, *x*)
Like **getyx( )**, these routines store the current beginning coordinates and size of the specified window.
Note: **getbegyx( )** and **getmaxyx( )** are macros.

**insch** (*ch*)
**winsch** (*win*, *ch*)
**mvwinsch** (*win*, *y*, *x*, *ch*)
**mvinsch** (*y*, *x*, *ch*)
The character *ch* is inserted before the character under the cursor. All characters to the right are moved one SPACE to the right, possibly losing the rightmost character of the line. The cursor position does not change (after moving to (*y*, *x*), if specified). (This does not imply use of the hardware "insert-character" feature.)
Note: *ch* is actually of type **chtype**, not a character.
Note: **insch( )**, **mvinsch( )**, and **mvwinsch( )** are macros.

**insertln( )**
**winsertln** (*win*)
A blank line is inserted above the current line and the bottom line is lost. (This does not imply use of the hardware "insert-line" feature.)
Note: **insertln( )** is a macro.

**move** (*y*, *x*)
**wmove** (*win*, *y*, *x*)
The cursor associated with the window is moved to line (row) *y*, column *x*. This does not move the physical cursor of the terminal until **refresh( )** is called. The position specified is relative to the upper left corner of the window, which is (**0**, **0**).
Note: **move( )** is a macro.

**overlay** (*srcwin*, *dstwin*)
**overwrite** (*srcwin*, *dstwin*)
These routines overlay *srcwin* on top of *dstwin*; that is, all text in *srcwin* is copied into *dstwin*. *scrwin* and *dstwin* need not be the same size; only text where the two windows overlap is copied. The difference is that **overlay( )** is non-destructive (blanks are not copied), while **overwrite( )** is destructive.

**copywin** (*srcwin*, *dstwin*, *sminrow*, *smincol*, *dminrow*, *dmincol*, *dmaxrow*, *dmaxcol*, *overlay*)
This routine provides a finer grain of control over the **overlay( )** and **overwrite( )** routines. Like in the **prefresh( )** routine, a rectangle is specified in the destination window, (*dminrow*, *dmincol*) and (*dmaxrow*, *dmaxcol*), and the upper-left-corner coordinates of the source window, (*sminrow*, *smincol*). If the argument *overlay* is true, then copying is non-destructive, as in **overlay( )**.

**printw** (*fmt* [, *arg* ...])
**wprintw** (*win*, *fmt* [, arg ...])
**mvprintw** (*y*, *x*, *fmt* [, *arg* ...])
**mvwprintw** (*win*, *y*, *x*, *fmt* [, *arg* ...])

These routines are analogous to **printf**(3S). The string that would be output by **printf**(3S) is instead output using **waddstr**( ) on the given window.

**vwprintw** (*win, fmt, varglist*)
This routine corresponds to **vprintf**(3V). It performs a **wprintw**( ) using a variable argument list. The third argument is a **va_list**, a pointer to a list of arguments, as defined in **<varargs.h>**. See the **vprintf**(3V) and **varargs**(3) manual pages for a detailed description on how to use variable argument lists.

**scroll** (*win*) The window is scrolled up one line. This involves moving the lines in the window data structure. As an optimization, if the window is **stdscr** and the scrolling region is the entire window, the physical screen will be scrolled at the same time.

**touchwin** (*win*)
**touchline** (*win, start, count*)
Throw away all optimization information about which parts of the window have been touched, by pretending that the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window, but the records of which lines have been changed in the other window will not reflect the change. **touchline**( ) only pretends that *count* lines have been changed, beginning with line *start*.

**Input**

**getch**( )
**wgetch** (*win*)
**mvgetch** (*y, x*)
**mvwgetch** (*win, y, x*) A character is read from the terminal associated with the window. In NODELAY mode, if there is no input waiting, the value **ERR** is returned. In DELAY mode, the program will hang until the system passes text through to the program. Depending on the setting of **cbreak**( ), this will be after one character (CBREAK mode), or after the first newline (NOCBREAK mode). In HALF-DELAY mode, the program will hang until a character is typed or the specified timeout has been reached. Unless **noecho**( ) has been set, the character will also be echoed into the designated window. No **refresh**( ) will occur between the **move**( ) and the **getch**( ) done within the routines **mvgetch**( ) and **mvwgetch**( ).
When using **getch**( ), **wgetch**( ), **mvgetch**( ), or **mvwgetch**( ), do not set both NOCBREAK mode (**nocbreak**( )) and ECHO mode (**echo**( )) at the same time. Depending on the state of the terminal driver when each character is typed, the program may produce undesirable results.
If **keypad** (*win,* **TRUE**) has been called, and a function key is pressed, the token for that function key will be returned instead of the raw characters. (See **keypad**( ) under **Input Options Setting**.) Possible function keys are defined in **<curses.h>** with integers beginning with **0401**, whose names begin with **KEY_**. If a character is received that could be the beginning of a function key (such as escape), **curses** will set a timer. If the remainder of the sequence is not received within the designated time, the character will be passed through, otherwise the function key value will be returned. For this reason, on many terminals, there will be a delay after a user presses the escape key before the escape is returned to the program. (Use by a programmer of the escape key for a single character routine is discouraged. Also see **notimeout**( ) below.)
Note: **getch**( ), **mvgetch**( ), and **mvwgetch**( ) are macros.

**getstr** (*str*)
**wgetstr** (*win, str*)
**mvgetstr** (*y, x, str*)
**mvwgetstr** (*win, y, x, str*)
A series of calls to **getch**( ) is made, until a newline, carriage return, or enter key

is received. The resulting value is placed in the area pointed at by the character pointer *str*. The user's erase and kill characters are interpreted. As in **mvgetch( )**, no **refresh( )** is done between the **move( )** and **getstr( )** within the routines **mvgetstr( )** and **mvwgetstr( )**.
Note: **getstr( )**, **mvgetstr( )**, and **mvwgetstr( )** are macros.

**flushinp( )** Throws away any typeahead that has been typed by the user and has not yet been read by the program.

**ungetch** (*c*) Place *c* back onto the input queue to be returned by the next call to **wgetch( )**.

**inch( )**
**winch** (*win*)
**mvinch** (*y, x*)
**mvwinch** (*win, y, x*) The character, of type **chtype**, at the current position in the named window is returned. If any attributes are set for that position, their values will be OR'ed into the value returned. The predefined constants **A_CHARTEXT** and **A_ATTRIBUTES**, defined in **<curses.h>**, can be used with the C logical AND (&) operator to extract the character or attributes alone.
Note: **inch( )**, **winch( )**, **mvinch( )**, and **mvwinch( )** are macros.

**scanw** (*fmt*[,*arg* . . .] )
**wscanw** (*win*,*fmt* [, *arg* . . .])
**mvscanw** (*y, x*,*fmt* [, *arg* . . .])
**mvwscanw** (*win, y, x*,*fmt* [, *arg* . . .])
These routines correspond to **scanf**(3V), as do their arguments and return values. **wgetstr( )** is called on the window, and the resulting line is used as input for the scan.

**vwscanw** (*win*,*fmt*, *ap*) This routine is similar to **vwprintw( )** above in that performs a **wscanw( )** using a variable argument list. The third argument is a **va_list**, a pointer to a list of arguments, as defined in **<varargs.h>**. See the **vprintf**(3V) and **varargs**(3) manual pages for a detailed description on how to use variable argument lists.

**Output Options Setting**

These routines set options within **curses** that deal with output. All options are initially FALSE, unless otherwise stated. It is not necessary to turn these options off before calling **endwin( )**.

**clearok** (*win, bf*) If enabled (*bf* is **TRUE**), the next call to **wrefresh( )** with this window will clear the screen completely and redraw the entire screen from scratch. This is useful when the contents of the screen are uncertain, or in some cases for a more pleasing visual effect.

**idlok** (*win, bf*) If enabled (*bf* is **TRUE**), **curses** will consider using the hardware "insert/delete-line" feature of terminals so equipped. If disabled (*bf* is **FALSE**), **curses** will very seldom use this feature. (The "insert/delete-character" feature is always considered.) This option should be enabled only if your application needs "insert/delete-line", for example, for a screen editor. It is disabled by default because "insert/delete-line" tends to be visually annoying when used in applications where it is not really needed. If "insert/delete-line" cannot be used, **curses** will redraw the changed portions of all lines.

**leaveok** (*win, bf*) Normally, the hardware cursor is left at the location of the window cursor being refreshed. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

**setscrreg** (*top, bot*)
**wsetscrreg** (*win, top, bot*)

These routines allow the user to set a software scrolling region in a window. *top* and *bot* are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and **scrollok()** are enabled, an attempt to move off the bottom margin line will cause all lines in the scrolling region to scroll up one line. (Note: this has nothing to do with use of a physical scrolling region capability in the terminal, like that in the DEC VT100. Only the text of the window is scrolled; if **idlok()** is enabled and the terminal has either a scrolling region or "insert/delete-line" capability, they will probably be used by the output routines.)
Note: **setscrreg()** and **wsetscrreg()** are macros.

**scrollok** (*win, bf*)

This option controls what happens when the cursor of a window is moved off the edge of the window or scrolling region, either from a newline on the bottom line, or typing the last character of the last line. If disabled (*bf* is **FALSE**), the cursor is left on the bottom line at the location where the offending character was entered. If enabled (*bf* is **TRUE**), **wrefresh()** is called on the window, and then the physical terminal and window are scrolled up one line. (Note: in order to get the physical scrolling effect on the terminal, it is also necessary to call **idlok()**.)

**nl()**
**nonl()**

These routines control whether NEWLINE is translated into RETURN and LINEFEED on output, and whether RETURN is translated into NEWLINE on input. Initially, the translations do occur. By disabling these translations using **nonl()**, **curses** is able to make better use of the linefeed capability, resulting in faster cursor motion.

**Input Options Setting**

These routines set options within **curses** that deal with input. The options involve using **ioctl**(2) and therefore interact with **curses** routines. It is not necessary to turn these options off before calling **endwin()**.
For more information on these options, refer to *Programming Utilities and Libraries*.

**cbreak()**
**nocbreak()**

These two routines put the terminal into and out of CBREAK mode, respectively. In CBREAK mode, characters typed by the user are immediately available to the program and erase/kill character processing is not performed. When in NOCBREAK mode, the tty driver will buffer characters typed until a NEWLINE or RETURN is typed. Interrupt and flow-control characters are unaffected by this mode (see **termio**(4)). Initially the terminal may or may not be in CBREAK mode, as it is inherited, therefore, a program should call **cbreak()** or **nocbreak()** explicitly. Most interactive programs using **curses** will set CBREAK mode.
Note: **cbreak()** overrides **raw()**. See **getch()** under **Input** for a discussion of how these routines interact with **echo()** and **noecho()**.

**echo()**
**noecho()**

These routines control whether characters typed by the user are echoed by **getch()** as they are typed. Echoing by the tty driver is always disabled, but initially **getch()** is in ECHO mode, so characters typed are echoed. Authors of most interactive programs prefer to do their own echoing in a controlled area of the screen, or not to echo at all, so they disable echoing by calling **noecho()**. See **getch()** under **Input** for a discussion of how these routines interact with **cbreak()** and **nocbreak()**.

**halfdelay** (*tenths*)

Half-delay mode is similar to CBREAK mode in that characters typed by the user are immediately available to the program. However, after blocking for *tenths* tenths of seconds, ERR will be returned if nothing has been typed. *tenths* must be a number between 1 and 255. Use **nocbreak()** to leave half-delay mode.

**intrflush** (*win, bf*) If this option is enabled, when an interrupt key is pressed on the keyboard (interrupt, break, quit) all output in the tty driver queue will be flushed, giving the effect of faster response to the interrupt, but causing **curses** to have the wrong idea of what is on the screen. Disabling the option prevents the flush. The default for the option is inherited from the tty driver settings. The window argument is ignored.

**keypad** (*win, bf*) This option enables the keypad of the user's terminal. If enabled, the user can press a function key (such as an arrow key) and **wgetch()** will return a single value representing the function key, as in **KEY_LEFT**. If disabled, **curses** will not treat function keys specially and the program would have to interpret the escape sequences itself. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option will cause the terminal keypad to be turned on when **wgetch()** is called.

**meta** (*win, bf*) If enabled, characters returned by **wgetch()** are transmitted with all 8 bits, instead of with the highest bit stripped. In order for **meta()** to work correctly, the **km** (**has_meta_key**) capability has to be specified in the terminal's **terminfo**(5V) entry.

**nodelay** (*win, bf*) This option causes **wgetch()** to be a non-blocking call. If no input is ready, **wgetch()** will return ERR. If disabled, **wgetch()** will hang until a key is pressed.

**notimeout** (*win, bf*) While interpreting an input escape sequence, **wgetch()** will set a timer while waiting for the next character. If **notimeout** (*win,* **TRUE**) is called, then **wgetch()** will not set a timer. The purpose of the timeout is to differentiate between sequences received from a function key and those typed by a user.

**raw()**
**noraw()** The terminal is placed into or out of RAW mode. RAW mode is similar to CBREAK mode, in that characters typed are immediately passed through to the user program. The differences are that in RAW mode, the interrupt, quit, suspend, and flow control characters are passed through uninterpreted, instead of generating a signal. RAW mode also causes 8-bit input and output. The behavior of the BREAK key depends on other bits in the terminal driver that are not set by **curses**.

**typeahead** (*fildes*) **curses** does "line-breakout optimization" by looking for typeahead periodically while updating the screen. If input is found, and it is coming from a tty, the current update will be postponed until **refresh()** or **doupdate()** is called again. This allows faster response to commands typed in advance. Normally, the file descriptor for the input FILE pointer passed to **newterm()**, or **stdin** in the case that **initscr()** was used, will be used to do this typeahead checking. The **typeahead()** routine specifies that the file descriptor *fildes* is to be used to check for typeahead instead. If *fildes* is –1, then no typeahead checking will be done.

Note: *fildes* is a file descriptor, not a **<stdio.h>** FILE pointer.

**Environment Queries**

**baudrate()** Returns the output speed of the terminal. The number returned is in bits per second, for example, 9600, and is an integer.

**char erasechar()** The user's current erase character is returned.

**has_ic()** True if the terminal has insert- and delete-character capabilities.

**has_il()** True if the terminal has insert- and delete-line capabilities, or can simulate them using scrolling regions. This might be used to check to see if it would be appropriate to turn on physical scrolling using **scrollok()**.

**char killchar()** The user's current line-kill character is returned.

**char *longname()** This routine returns a pointer to a static area containing a verbose description of

the current terminal. The maximum length of a verbose description is 128 characters. It is defined only after the call to **initscr()** or **newterm()**. The area is overwritten by each call to **newterm()** and is not restored by **set_term()**, so the value should be saved between calls to **newterm()** if **longname()** is going to be used with multiple terminals.

**Soft Labels**

If desired, **curses** will manipulate the set of soft function-key labels that exist on many terminals. For those terminals that do not have soft labels, if you want to simulate them, **curses** will take over the bottom line of **stdscr**, reducing the size of **stdscr** and the variable LINES. **curses** standardizes on 8 labels of 8 characters each.

**slk_init** (*labfmt*) In order to use soft labels, this routine must be called before **initscr()** or **newterm()** is called. If **initscr()** winds up using a line from **stdscr** to emulate the soft labels, then *labfmt* determines how the labels are arranged on the screen. Setting *labfmt* to **0** indicates that the labels are to be arranged in a 3-2-3 arrangement; **1** asks for a 4-4 arrangement.

**slk_set** (*labnum, label, labfmt*)
*labnum* is the label number, from 1 to 8. *label* is the string to be put on the label, up to 8 characters in length. A NULL string or a NULL pointer will put up a blank label. *labfmt* is one of **0**, **1** or **2**, to indicate whether the label is to be left-justified, centered, or right-justified within the label.

**slk_refresh()**
**slk_noutrefresh()** These routines correspond to the routines **wrefresh()** and **wnoutrefresh()**. Most applications would use **slk_noutrefresh()** because a **wrefresh()** will most likely soon follow.

**char *slk_label** (*labnum*)
The current label for label number *labnum*, with leading and trailing blanks stripped, is returned.

**slk_clear()** The soft labels are cleared from the screen.

**slk_restore()** The soft labels are restored to the screen after a **slk_clear()**.

**slk_touch()** All of the soft labels are forced to be output the next time a **slk_noutrefresh()** is performed.

**Low-Level curses Access**

The following routines give low-level access to various **curses** functionality. These routines typically would be used inside of library routines.

**def_prog_mode()**
**def_shell_mode()** Save the current terminal modes as the "program" (in **curses**) or "shell" (not in **curses**) state for use by the **reset_prog_mode()** and **reset_shell_mode()** routines. This is done automatically by **initscr()**.

**reset_prog_mode()**
**reset_shell_mode()** Restore the terminal to "program" (in **curses**) or "shell" (out of **curses**) state. These are done automatically by **endwin()** and **doupdate()** after an **endwin()**, so they normally would not be called.

**resetty()**
**savetty()** These routines save and restore the state of the terminal modes. **savetty()** saves the current state of the terminal in a buffer and **resetty()** restores the state to what it was at the last call to **savetty()**.

**getsyx** (*y, x*) The current coordinates of the virtual screen cursor are returned in *y* and *x*. Like **getyx()**, the variables *y* and *x* do not take an **&** before them. If **leaveok()** is

currently TRUE, then **−1**, **−1** will be returned. If lines may have been removed from the top of the screen using **ripoffline()** and the values are to be used beyond just passing them on to **setsyx()**, the value *y*+**stdscr−>_yoffset** should be used for those other uses.

Note: **getsyx()** is a macro.

**setsyx** (*y*, *x*) The virtual screen cursor is set to *y*, *x*. If *y* and *x* are both **−1**, then **leaveok()** will be set. The two routines **getsyx()** and **setsyx()** are designed to be used by a library routine that manipulates **curses** windows but does not want to mess up the current position of the program's cursor. The library routine would call **getsyx()** at the beginning, do its manipulation of its own windows, do a **wnoutrefresh()** on its windows, call **setsyx()**, and then call **doupdate()**.

**ripoffline** (*line, init*) This routine provides access to the same facility that **slk_init()** uses to reduce the size of the screen. **ripoffline()** must be called before **initscr()** or **newterm()** is called. If *line* is positive, a line will be removed from the top of **stdscr**; if negative, a line will be removed from the bottom. When this is done inside **initscr()**, the routine *init* is called with two arguments: a window pointer to the 1-line window that has been allocated and an integer with the number of columns in the window. Inside this initialization routine, the integer variables **LINES** and **COLS** (defined in **<curses.h>**) are not guaranteed to be accurate and **wrefresh()** or **doupdate()** must not be called. It is allowable to call **wnoutrefresh()** during the initialization routine.

**ripoffline()** can be called up to five times before calling **initscr()** or **newterm()**.

**scr_dump** (*filename*) The current contents of the virtual screen are written to the file *filename*.

**scr_restore** (*filename*) The virtual screen is set to the contents of *filename*, which must have been written using **scr_dump()**. The next call to **doupdate()** will restore the screen to what it looked like in the dump file.

**scr_init** (*filename*) The contents of *filename* are read in and used to initialize the **curses** data structures about what the terminal currently has on its screen. If the data is determined to be valid, **curses** will base its next update of the screen on this information rather than clearing the screen and starting from scratch. **scr_init()** would be used after **initscr()** or a **system**(3) call to share the screen with another process that has done a **scr_dump()** after its **endwin()** call. The data will be declared invalid if the time-stamp of the tty is old or the **terminfo**(5V) capability **nrrmc** is true.

**curs_set** (*visibility*) The cursor is set to invisible, normal, or very visible for *visibility* equal to **0**, **1** or **2**.

**draino** (*ms*) Wait until the output has drained enough that it will only take *ms* more milliseconds to drain completely.

**garbagedlines** (*win, begline, numlines*)
This routine indicates to **curses** that a screen line is garbaged and should be thrown away before having anything written over the top of it. It could be used for programs such as editors that want a command to redraw just a single line. Such a command could be used in cases where there is a noisy communications line and redrawing the entire screen would be subject to even more communication noise. Just redrawing the single line gives some semblance of hope that it would show up unblemished. The current location of the window is used to determine which lines are to be redrawn.

**napms** (*ms*) Sleep for *ms* milliseconds.

**Terminfo-Level Manipulations**

These low-level routines must be called by programs that need to deal directly with the **terminfo**(5V) database to handle certain terminal capabilities, such as programming function keys. For all other functionality, **curses** routines are more suitable and their use is recommended.

Initially, **setupterm**( ) should be called. (Note: **setupterm**( ) is automatically called by **initscr**( ) and **newterm**( ).) This will define the set of terminal-dependent variables defined in the **terminfo**(5V) database. The **terminfo**(5V) variables *lines* and *columns* (see **terminfo**(5V)) are initialized by **setupterm**( ) as follows: if the environment variables **LINES** and **COLUMNS** exist, their values are used. If the above environment variables do not exist, and the window sizes in rows and columns as returned by the **TIOCGWINSZ ioctl** are non-zero, those sizes are used. Otherwise, the values for *lines* and *columns* specified in the **terminfo**(5V) database are used.

The header files **<curses.h>** and **<term.h>** should be included, in this order, to get the definitions for these strings, numbers, and flags. Parameterized strings should be passed through **tparm**( ) to instantiate them. All **terminfo**(5V) strings (including the output of **tparm**( ) should be printed with **tputs**( ) or **putp**( ). Before exiting, **reset_shell_mode**( ) should be called to restore the tty modes. Programs that use cursor addressing should output **enter_ca_mode** upon startup and should output **exit_ca_mode** before exiting (see **terminfo**(5V)). (Programs desiring shell escapes should call **reset_shell_mode**( ) and output **exit_ca_mode** before the shell is called and should output **enter_ca_mode** and call **reset_prog_mode**( ) after returning from the shell. Note: this is different from the **curses** routines (see **endwin**( ))

**setupterm** (*term, fildes, errret*)
Reads in the **terminfo**(5V) database, initializing the **terminfo**(5V) structures, but does not set up the output virtualization structures used by **curses**. The terminal type is in the character string *term*; if *term* is NULL, the environment variable **TERM** will be used. All output is to the file descriptor *fildes*. If *errret* is not NULL, then **setupterm**( ) will return **OK** or **ERR** and store a status value in the integer pointed to by *errret*. A status of **1** in *errret* is normal, **0** means that the terminal could not be found, and **−1** means that the **terminfo**(5V) database could not be found. If *errret* is NULL, **setupterm**( ) will print an error message upon finding an error and exit. Thus, the simplest call is '**setupterm ((char *)0, 1, (int *)0)**', which uses all the defaults.

The **terminfo**(5V) boolean, numeric and string variables are stored in a structure of type **TERMINAL**. After **setupterm**( ) returns successfully, the variable *cur_term* (of type **TERMINAL ***) is initialized with all of the information that the **terminfo**(5V) boolean, numeric and string variables refer to. The pointer may be saved before calling **setupterm**( ) again. Further calls to **setupterm**( ) will allocate new space rather than reuse the space pointed to by *cur_term*.

**set_curterm** (*nterm*) *nterm* is of type **TERMINAL *** . **set_curterm**( ) sets the variable *cur_term* to *nterm*, and makes all of the **terminfo**(5V) boolean, numeric and string variables use the values from *nterm*.

**del_curterm** (*oterm*) *oterm* is of type **TERMINAL ***. **del_curterm**( ) frees the space pointed to by *oterm* and makes it available for further use. If *oterm* is the same as *cur_term*, then references to any of the **terminfo**(5V) boolean, numeric and string variables thereafter may refer to invalid memory locations until another **setupterm**( ) has been called.

**restartterm** (*term, fildes, errret*)
Like **setupterm**( ) after a memory restore.

**char *tparm** (*str*, $p_1$, $p_2$, ..., $p_9$)
Instantiate the string *str* with parms $p_i$. A pointer is returned to the result of *str* with the parameters applied.

**tputs** (*str, count, putc*) Apply padding to the string *str* and output it. *str* must be a **terminfo**(5V) string

variable or the return value from **tparm()**, **tgetstr()**, **tigetstr()** or **tgoto()**. *count* is the number of lines affected, or 1 if not applicable. **putchar()** is a **putc**(3S)-like routine to which the characters are passed, one at a time.

**putp** (*str*) A routine that calls **tputs()** (*str*, **1**, **putchar()**).

**vidputs** (*attrs, putc*) Output a string that puts the terminal in the video attribute mode *attrs*, which is any combination of the attributes listed below. The characters are passed to the **putc**(3S)-like routine **putchar()**.

**vidattr** (*attrs*) Like **vidputs()**, except that it outputs through **putchar**(3S).

**mvcur** (*oldrow, oldcol, newrow, newcol*)
Low-level cursor motion.

The following routines return the value of the capability corresponding to the **terminfo**(5V) *capname* passed to them, such as **xenl**.

**tigetflag** (*capname*) The value **−1** is returned if *capname* is not a boolean capability.

**tigetnum** (*capname*) The value **−2** is returned if *capname* is not a numeric capability.

**tigetstr** (*capname*) The value (**char ***) **−1** is returned if *capname* is not a string capability.

**char *boolnames**[ ], ***boolcodes**[ ], ***boolfnames**[ ]
**char *numnames**[ ], ***numcodes**[ ], ***numfnames**[ ]
**char *strnames**[ ], ***strcodes**[ ], ***strfnames**[ ]
These null-terminated arrays contain the *capnames*, the **termcap**(5) codes, and the full C names, for each of the **terminfo**(5V) variables.

**Termcap Emulation**

These routines are included as a conversion aid for programs that use the **termcap**(3X) library. Their parameters are the same and the routines are emulated using the **terminfo**(5V) database.

**tgetent** (*bp, name*) Look up **termcap** entry for *name*. The emulation ignores the buffer pointer *bp*.

**tgetflag** (*codename*) Get the boolean entry for *codename*.

**tgetnum** (*codes*) Get numeric entry for *codename*.

**char *tgetstr** (*codename, area*)
Return the string entry for *codename*. If *area* is not NULL, then also store it in the buffer pointed to by *area* and advance *area*. **tputs()** should be used to output the returned string.

**char *tgoto** (*cap, col, row*)
Instantiate the parameters into the given capability. The output from this routine is to be passed to **tputs()**.

**tputs** (*str, affcnt, putc*) See **tputs()** above, under **Terminfo-Level Manipulations**.

**Miscellaneous**

**unctrl** (*c*) This macro expands to a character string which is a printable representation of the character *c*. Control characters are displayed in the ^X notation. Printing characters are displayed as is.

**unctrl()** is a macro, defined in **<unctrl.h>**, which is automatically included by **<curses.h>**.

**char *keyname** (*c*) A character string corresponding to the key *c* is returned.

**filter()** This routine is one of the few that is to be called before **initscr()** or **newterm()** is called. It arranges things so that **curses** thinks that there is a 1-line screen. **curses** will not use any terminal capabilities that assume that they know what line on the screen the cursor is on.

**Use of curscr**

The special window **curscr** can be used in only a few routines. If the window argument to **clearok( )** is **curscr**, the next call to **wrefresh( )** with any window will cause the screen to be cleared and repainted from scratch. If the window argument to **wrefresh( )** is **curscr**, the screen is immediately cleared and repainted from scratch. (This is how most programs would implement a "repaint-screen" routine.) The source window argument to **overlay( )**, **overwrite( )**, and **copywin** may be **curscr**, in which case the current contents of the virtual terminal screen will be accessed.

**Obsolete Calls**

Various routines are provided to maintain compatibility in programs written for older versions of the curses library. These routines are all emulated as indicated below.

| | |
|---|---|
| **crmode( )** | Replaced by **cbreak( )**. |
| **fixterm( )** | Replaced by **reset_prog_mode( )**. |
| **gettmode( )** | A no-op. |
| **nocrmode( )** | Replaced by **nocbreak( )**. |
| **resetterm( )** | Replaced by **reset_shell_mode( )**. |
| **saveterm( )** | Replaced by **def_prog_mode( )**. |
| **setterm( )** | Replaced by **setupterm( )**. |

ATTRIBUTES

The following video attributes, defined in **<curses.h>**, can be passed to the routines **attron( )**, **attroff( )**, and **attrset( )**, or OR'ed with the characters passed to **addch( )**.

| | |
|---|---|
| A_STANDOUT | Terminal's best highlighting mode |
| A_UNDERLINE | Underlining |
| A_REVERSE | Reverse video |
| A_BLINK | Blinking |
| A_DIM | Half bright |
| A_BOLD | Extra bright or bold |
| A_ALTCHARSET | Alternate character set |
| | |
| A_CHARTEXT | Bit-mask to extract character (described under **winch**) |
| A_ATTRIBUTES | Bit-mask to extract attributes (described under **winch**) |
| A_NORMAL | Bit mask to reset all attributes off (for example: '**attrset (A_NORMAL)**' |

FUNCTION-KEYS

The following function keys, defined in **<curses.h>**, might be returned by **getch( )** if **keypad( )** has been enabled. Note: not all of these may be supported on a particular terminal if the terminal does not transmit a unique code when the key is pressed or the definition for the key is not present in the **terminfo**(5V) database.

| *Name* | *Value* | *Key name* |
|---|---|---|
| KEY_BREAK | 0401 | break key (unreliable) |
| KEY_DOWN | 0402 | The four arrow keys ... |
| KEY_UP | 0403 | |
| KEY_LEFT | 0404 | |
| KEY_RIGHT | 0405 | ... |
| KEY_HOME | 0406 | Home key (upward+left arrow) |
| KEY_BACKSPACE | 0407 | backspace (unreliable) |
| KEY_F0 | 0410 | Function keys. Space for 64 keys is reserved. |
| KEY_F(n) | (KEY_F0+(n)) | Formula for $f_n$. |
| KEY_DL | 0510 | Delete line |
| KEY_IL | 0511 | Insert line |
| KEY_DC | 0512 | Delete character |

| | | |
|---|---|---|
| KEY_IC | 0513 | Insert char or enter insert mode |
| KEY_EIC | 0514 | Exit insert char mode |
| KEY_CLEAR | 0515 | Clear screen |
| KEY_EOS | 0516 | Clear to end of screen |
| KEY_EOL | 0517 | Clear to end of line |
| KEY_SF | 0520 | Scroll 1 line forward |
| KEY_SR | 0521 | Scroll 1 line backwards (reverse) |
| KEY_NPAGE | 0522 | Next page |
| KEY_PPAGE | 0523 | Previous page |
| KEY_STAB | 0524 | Set TAB |
| KEY_CTAB | 0525 | Clear TAB |
| KEY_CATAB | 0526 | Clear all TAB characters |
| KEY_ENTER | 0527 | Enter or send |
| KEY_SRESET | 0530 | soft (partial) reset |
| KEY_RESET | 0531 | reset or hard reset |
| KEY_PRINT | 0532 | print or copy |
| KEY_LL | 0533 | home down or bottom (lower left) |
| | | keypad is arranged like this: |
| | | A1 up A3 |
| | | left B2 right |
| | | C1 down C3 |
| KEY_A1 | 0534 | Upper left of keypad |
| KEY_A3 | 0535 | Upper right of keypad |
| KEY_B2 | 0536 | Center of keypad |
| KEY_C1 | 0537 | Lower left of keypad |
| KEY_C3 | 0540 | Lower right of keypad |
| KEY_BTAB | 0541 | Back TAB key |
| KEY_BEG | 0542 | beg(inning) key |
| KEY_CANCEL | 0543 | cancel key |
| KEY_CLOSE | 0544 | close key |
| KEY_COMMAND | 0545 | cmd (command) key |
| KEY_COPY | 0546 | copy key |
| KEY_CREATE | 0547 | create key |
| KEY_END | 0550 | end key |
| KEY_EXIT | 0551 | exit key |
| KEY_FIND | 0552 | find key |
| KEY_HELP | 0553 | help key |
| KEY_MARK | 0554 | mark key |
| KEY_MESSAGE | 0555 | message key |
| KEY_MOVE | 0556 | move key |
| KEY_NEXT | 0557 | next object key |
| KEY_OPEN | 0560 | open key |
| KEY_OPTIONS | 0561 | options key |
| KEY_PREVIOUS | 0562 | previous object key |
| KEY_REDO | 0563 | redo key |
| KEY_REFERENCE | 0564 | ref(erence) key |
| KEY_REFRESH | 0565 | refresh key |
| KEY_REPLACE | 0566 | replace key |
| KEY_RESTART | 0567 | restart key |
| KEY_RESUME | 0570 | resume key |
| KEY_SAVE | 0571 | save key |
| KEY_SBEG | 0572 | shifted beginning key |
| KEY_SCANCEL | 0573 | shifted cancel key |

| | | |
|---|---|---|
| KEY_SCOMMAND | 0574 | shifted command key |
| KEY_SCOPY | 0575 | shifted copy key |
| KEY_SCREATE | 0576 | shifted create key |
| KEY_SDC | 0577 | shifted delete char key |
| KEY_SDL | 0600 | shifted delete line key |
| KEY_SELECT | 0601 | select key |
| KEY_SEND | 0602 | shifted end key |
| KEY_SEOL | 0603 | shifted clear line key |
| KEY_SEXIT | 0604 | shifted exit key |
| KEY_SFIND | 0605 | shifted find key |
| KEY_SHELP | 0606 | shifted help key |
| KEY_SHOME | 0607 | shifted home key |
| KEY_SIC | 0610 | shifted input key |
| KEY_SLEFT | 0611 | shifted left arrow key |
| KEY_SMESSAGE | 0612 | shifted message key |
| KEY_SMOVE | 0613 | shifted move key |
| KEY_SNEXT | 0614 | shifted next key |
| KEY_SOPTIONS | 0615 | shifted options key |
| KEY_SPREVIOUS | 0616 | shifted prev key |
| KEY_SPRINT | 0617 | shifted print key |
| KEY_SREDO | 0620 | shifted redo key |
| KEY_SREPLACE | 0621 | shifted replace key |
| KEY_SRIGHT | 0622 | shifted right arrow |
| KEY_SRSUME | 0623 | shifted resume key |
| KEY_SSAVE | 0624 | shifted save key |
| KEY_SSUSPEND | 0625 | shifted suspend key |
| KEY_SUNDO | 0626 | shifted undo key |
| KEY_SUSPEND | 0627 | suspend key |
| KEY_UNDO | 0630 | undo key |

**LINE GRAPHICS**

The following variables may be used to add line-drawing characters to the screen with **waddce**. When defined for the terminal, the variable will have the **A_ALTCHARSET** bit turned on. Otherwise, the default character listed below will be stored in the variable. The names were chosen to be consistent with the DEC VT100 nomenclature.

| *Name* | *Default* | *Glyph Description* |
|---|---|---|
| ACS_ULCORNER | + | upper left corner |
| ACS_LLCORNER | + | lower left corner |
| ACS_URCORNER | + | upper right corner |
| ACS_LRCORNER | + | lower right corner |
| ACS_RTEE | + | right tee (┤) |
| ACS_LTEE | + | left tee (├) |
| ACS_BTEE | + | bottom tee (┴) |
| ACS_TTEE | + | top tee (┬) |
| ACS_HLINE | – | horizontal line |
| ACS_VLINE | \| | vertical line |
| ACS_PLUS | + | plus |
| ACS_S1 | – | scan line 1 |
| ACS_S9 | _ | scan line 9 |
| ACS_DIAMOND | + | diamond |
| ACS_CKBOARD | : | checker board (stipple) |
| ACS_DEGREE | ' | degree symbol |
| ACS_PLMINUS | # | plus/minus |

| | | |
|---|---|---|
| ACS_BULLET | o | bullet |
| ACS_LARROW | < | arrow pointing left |
| ACS_RARROW | > | arrow pointing right |
| ACS_DARROW | v | arrow pointing down |
| ACS_UARROW | ^ | arrow pointing up |
| ACS_BOARD | # | board of squares |
| ACS_LANTERN | # | lantern symbol |
| ACS_BLOCK | # | solid square block |

RETURN VALUES

All routines return the integer **OK** upon successful completion and the integer **ERR** upon failure, unless otherwise noted in the preceding routine descriptions.

All macros return the value of their **w** version, except **setscrreg()**, **wsetscrreg()**, **getsyx()**, **getyx()**, **getbegy()**, **getmaxyx()**. For these macros, no useful value is returned.

Routines that return pointers always return (*type* *) **NULL** "on error.

FILES

**/usr/share/lib/terminfo**
**.login**
**.profile**

SEE ALSO

**cc**(1V), **ld**(1), **ioctl**(2), **plot**(3X), **printf**(3S), **putc**(3S), **scanf**(3V), **stdio**(3V), **system**(3), **varargs**(3), **vprintf**(3V), **termio**(4), **term(5V)**, **terminfo**(5V)

WARNINGS

The plotting library **plot**(3X) and the curses library **curses**(3V) both use the names **erase()** and **move()**. The **curses** versions are macros. If you need both libraries, put the **plot**(3X) code in a different source file than the **curses**(3V) code, and/or '**#undef move**' and '**#undef erase**' in the **plot**(3X) code.

Between the time a call to **initscr()** and **endwin()** has been issued, use only the routines in the **curses** library to generate output. Using system calls or the "standard I/O package" (see **stdio**(3V)) for output during that time can cause unpredictable results.

## NAME

ferror, feof, clearerr, fileno – stream status inquiries

## SYNOPSIS

```
#include <stdio.h>

ferror(stream)
FILE *stream;

feof(stream)
FILE *stream;

clearerr(stream)
FILE *stream;

fileno(stream)
FILE *stream;
```

## DESCRIPTION

**ferror**( ) returns non-zero when an error has occurred reading from or writing to the named stream, otherwise zero. Unless cleared by **clearerr**, the error indication lasts until the stream is closed.

**feof**( ) returns non-zero when EOF has previously been detected reading the named input stream, otherwise zero. Unless cleared by **clearerr**, the EOF indication lasts until the stream is closed; however, operations which attempt to read from the stream will ignore the current state of the EOF indication and attempt to read from the file descriptor associated with the stream.

**clearerr**( ) resets the error indication and EOF indication to zero on the named stream.

**fileno**( ) returns the integer file descriptor associated with the stream; see **open**(2V).

## NOTE

All these functions are implemented as macros; they cannot be redeclared.

## SEE ALSO

**open**(2V), **fopen**(3S)

## NAME

fopen, freopen, fdopen – open a stream

## SYNOPSIS

```
#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;
```

## DESCRIPTION

**fopen()** opens the file named by *filename* and associates a stream with it. If the open succeeds, **fopen()** returns a pointer to be used to identify the stream in subsequent operations.

*filename* points to a character string that contains the name of the file to be opened.

*type* is a character string having one of the following values:

| | |
|---|---|
| **r** | open for reading |
| **w** | truncate or create for writing |
| **a** | append: open for writing at end of file, or create for writing |
| **r+** | open for update (reading and writing) |
| **w+** | truncate or create for update |
| **a+** | append; open or create for update at EOF |

**freopen()** opens the file named by *filename* and associates the stream pointed to by *stream* with it. The *type* argument is used just as in **fopen**. The original stream is closed, regardless of whether the open ultimately succeeds. If the open succeeds, **freopen()** returns the original value of *stream*.

**freopen()** is typically used to attach the preopened streams associated with **stdin**, **stdout**, and **stderr** to other files.

**fdopen()** associates a stream with the file descriptor *fildes*. File descriptors are obtained from calls like **open**, **dup**, **creat**, or **pipe**(2), which open files but do not return streams. Streams are necessary input for many of the Section 3S library routines. The *type* of the stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening **fseek()** or **rewind**, and input may not be directly followed by output without an intervening **fseek**, **rewind**, or an input operation which encounters end-of-file.

When a file is opened for append (that is, when *type* is **a** or **a+**), it is impossible to overwrite information already in the file. **fseek()** may be used to reposition the file pointer to any position in the file, but when output is written to the file, the current file pointer is disregarded. All output is written at the end of the file and causes the file pointer to be repositioned at the end of the output. If two separate processes open the same file for append, each process may write freely to the file without fear of destroying output being written by the other. The output from the two processes will be intermixed in the file in the order in which it is written.

## SEE ALSO

**open**(2V), **pipe**(2), **fclose**(3S), **fopen**(3S), **fseek**(3S)

## DIAGNOSTICS

**fopen**, **freopen**, and **fdopen()** return a NULL pointer on failure.

**BUGS**

In order to support the same number of open files that the system does, **fopen()** must allocate additional memory for data structures using **calloc()** after 64 files have been opened. This confuses some programs which use their own memory allocators.

## NAME

getc, getchar, fgetc, getw – get character or integer from stream

## SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar( )

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;
```

## DESCRIPTION

**getc( )** returns the next character (that is, byte) from the named input stream, as an integer. It also moves the file pointer, if defined, ahead one character in stream. **getchar( )** is defined as **getc(stdin)**. **getc** and **getchar** are macros.

**fgetc( )** behaves like **getc**, but is a function rather than a macro. **fgetc( )** runs more slowly than **getc**, but it takes less space per invocation and its name can be passed as an argument to a function.

**getw( )** returns the next C **int** (*word*) from the named input stream. **getw( )** increments the associated file pointer, if defined, to point to the next word. The size of a word is the size of an integer and varies from machine to machine. **getw( )** assumes no special alignment in the file.

## SEE ALSO

**ferror**(3S), **fopen**(3S), **fread**(3S), **gets**(3S), **putc**(3S), **scanf**(3S), **ungetc**(3S)

## DIAGNOSTICS

These functions return the integer constant EOF at EOF or upon an error. Because EOF is a valid integer, **ferror**(3S) should be used to detect **getw( )** errors.

## WARNING

If the integer value returned by **getc**, **getchar**, or **fgetc** is stored into a character variable and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a character on widening to integer is machine-dependent.

## BUGS

Because it is implemented as a macro, **getc( )** treats a stream argument with side effects incorrectly. In particular, **getc(*f++)** does not work sensibly. **fgetc( )** should be used instead.

Because of possible differences in word length and byte ordering, files written using **putw( )** are machine-dependent, and may not be readable using **getw( )** on a different processor.

NAME

getpass – read a password

SYNOPSIS

**char *getpass(prompt)**
**char *prompt;**

DESCRIPTION

**getpass( )** reads up to a NEWLINE or EOF from the file **/dev/tty**, after prompting with the NULL-terminated string *prompt* and disabling echoing. A pointer is returned to a NULL-terminated string of at most 8 characters. An interrupt will terminate input and send an interrupt signal to the calling program before returning. If **/dev/tty** cannot be opened, a NULL pointer is returned; the standard input is not read.

FILES

**/dev/tty**

SEE ALSO

**crypt**(3), **getpass**(3)

WARNING

The above routine uses **<stdio.h>**, which increases the size of programs not otherwise using standard I/O, more than might be expected.

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, setpwfile, fgetpwent – get password file entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent()

int endpwent()

setpwfile(name)
char *name;

struct passwd *fgetpwent(f)
FILE *f;
```

DESCRIPTION

**getpwent**, **getpwuid()** and **getpwnam()** each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file. Each line in the file contains a "passwd" structure, declared in the **<pwd.h>** header file:

```
struct   passwd { /* see getpwent(3) */
         char    *pw_name;
         char    *pw_passwd;
         int     pw_uid;
         int     pw_gid;
         char    *pw_age;
         char    *pw_comment;
         char    *pw_gecos;
         char    *pw_dir;
         char    *pw_shell;
};
struct passwd *getpwent(), *getpwuid(), *getpwnam();
```

This structure is declared in **<pwd.h>** so it is not necessary to redeclare it.

The field **pw_comment** is unused; the others have meanings described in **passwd**(5). When first called, **getpwent()** returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. **getpwuid()** searches from the beginning of the file until a numerical user ID matching *uid* is found and returns a pointer to the particular structure in which it was found. **getpwnam()** searches from the beginning of the file until a login name matching *name* is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to **getpwent()** has the effect of rewinding the password file to allow repeated searches. **endpwent()** may be called to close the password file when processing is complete.

**setpwfile()** changes the default password file to *name* thus allowing alternate password files to be used. Note: it does *not* close the previous file. If this is desired, **endpwent()** should be called prior to it.

**fgetpwent()** returns a pointer to the next passwd structure in the stream *f*, which matches the format of the password file **/etc/passwd**.

The field **pw_age** is used to hold a value for "password aging" on some systems; "password aging" is not supported on Sun systems. As such, it is effectively not used.

FILES

**/etc/passwd**
**/var/yp/***domainname***/passwd.byname**
**/var/yp/***domainname***/passwd.byuid**

SEE ALSO

**getgrent**(3), **getlogin**(3), **getpwent**(3), **passwd**(5), **ypserv**(8)

DIAGNOSTICS

A NULL pointer is returned on end-of-file or error.

WARNING

The above routines use the standard I/O library, which causes them to increase the size of programs, not otherwise using standard I/O, more than might be expected.

BUGS

All information is contained in a static area, so it must be copied if it is to be saved.

**NAME**

nice – change priority of a process

**SYNOPSIS**

**int nice(incr)**

**DESCRIPTION**

The scheduling priority of the process is augmented by *incr*. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without undue impact on system performance.

Negative increments are illegal, except when specified by the super-user. The priority is limited to the range –20 (most urgent) to 19 (least). Requests for values above or below these limits result in the scheduling priority being set to the corresponding limit.

The priority of a process is passed to a child process by **fork**(2).

**RETURN VALUE**

Upon successful completion, **nice()** returns the new scheduling priority. Otherwise, a value of –1 is returned and **errno** is set to indicate the error.

**ERRORS**

The priority is not changed if:

EPERM — The value of *incr* specified was negative, or greater than 40, and the effective user ID is not super-user.

**SEE ALSO**

**nice**(1), **fork**(2), **getpriority**(2), **renice**(8)

## NAME

nlist – get entries from symbol table

## SYNOPSIS

```
#include <nlist.h>

int nlist(filename, nl)
char *filename;
struct nlist *nl;
```

## DESCRIPTION

**nlist()** examines the symbol table from the executable image whose name is pointed to by *filename*, and selectively extracts a list of values and puts them in the array of **nlist()** structures pointed to by *nl*. The name list pointed to by **nl()** consists of an array of structures containing names, types and values. The *n_name* field of each such structure is taken to be a pointer to a character string representing a symbol name. The list is terminated by an entry with a NULL pointer (or a pointer to a NULL string) in the *n_name* field. For each entry in *nl*, if the named symbol is present in the executable image's symbol table, its value and type are placed in the *n_value* and *n_type* fields. If a symbol cannot be located, the corresponding *n_type* field of **nl()** is set to zero.

## RETURN VALUE

Upon normal completion, **nlist()** returns 0. If an error occurs, **nlist()** returns −1 and sets all of the *n_type* fields in members of the array pointed to by **nl()** to zero.

## SEE ALSO

**a.out**(5)

NAME

printf, fprintf, sprintf – formatted output conversion

SYNOPSIS

```
#include <stdio.h>
int printf(format [ , arg ] ... )
char *format;

int fprintf(stream, format [ , arg ] ... )
FILE *stream;
char *format;

int sprintf(s, format [ , arg ] ... )
char *s, *format;

#include <varargs.h>
int _doprnt(format, args, stream)
char *format;
va_list args;
FILE *stream;
```

DESCRIPTION

**printf()** places output on the standard output stream **stdout**. **fprintf()** places output on the named output **stream**. **sprintf()** places "output", followed by the NULL character (**\0**), in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough storage is available. **printf, fprintf()** and **sprintf()** return the number of characters transmitted (excluding the NULL character in the case of **sprintf**).

If an output error is encountered **printf, fprintf()** and **sprintf()** return EOF.

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of zero or more *arg*s. The results are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character **%**. After the **%**, the following appear in sequence:

Zero or more *flags*, which modify the meaning of the conversion specification.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag '–', described below, has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

A *precision* that gives the minimum number of digits to appear for the **d, i, o, u, x**, or **X** conversions, the number of digits to appear after the decimal point for the **e, E**, and **f** conversions, the maximum number of significant digits for the **g** and **G** conversion, or the maximum number of characters to be printed from a string in **s** conversion. The precision takes the form of a period (.) followed by a decimal digit string; a NULL digit string is treated as zero. Padding specified by the precision overrides the padding specified by the field width.

An optional **l** (ell) specifying that a following **d, i, o, u, x**, or **X** conversion character applies to a long integer *arg*. An **l** before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision or both may be indicated by an asterisk (*****) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted. A negative field width argument is taken as a '–' flag followed by a positive field width. If the precision argument is negative, it will be changed to zero.

The flag characters and their meanings are:

– The result of the conversion will be left-justified within the field.

+ The result of a signed conversion will always begin with a sign (+ or –).

blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

# This flag specifies that the value is to be converted to an "alternate form."For **c**, **d**, **i**, **s**, and **u** conversions, the flag has no effect. For **o** conversion, it increases the precision to force the first digit of the result to be a zero. For **x or X** conversion, a non-zero result will have **0x or 0X** prefixed to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For **g** and **G** conversions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

**d,i,o,u,x,X** The integer *arg* is converted to signed decimal (**d** or **i**), unsigned octal (**o**), unsigned decimal (**u**), or unsigned hexadecimal notation (**x** and **X**), respectively; the letters **abcdef** are used for **x** conversion and the letters **ABCDEF** for **X** conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prepending a zero to the field width. This does not imply an octal value for the field width.) The default precision is 1. The result of converting a zero value with a precision of zero is a NULL string.

**f** The float or double *arg* is converted to decimal notation in the style "[–]ddd.ddd" where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

**e,E** The float or double *arg* is converted in the style "[–]d.ddde±ddd," where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. The exponent always contains at least two digits.

**g,G** The float or double *arg* is printed in style **f** or **e** (or in style **E** in the case of a **G** format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style **e** or **E** will be used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

The **e**, **E**, **f**, **g**, and **G** formats print IEEE indeterminate values (infinity or not-a-number) as "Infinity" or "NaN" respectively.

**c** The character *arg* is printed.

**s** The *arg* is taken to be a string (character pointer) and characters from the string are printed until a NULL character (\0) is encountered or until the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first NULL character are printed. A NULL value for *arg* will yield undefined results.

**%** Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Padding takes place only if the specified field width exceeds the actual width. Characters generated by **printf**() and **fprintf**() are printed as if **putc**(3S) had been called.

## EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02," where *weekday* and *month* are pointers to NULL-terminated strings:

**printf("%s, %s %i, %d:%.2d", weekday, month, day, hour, min);**

To print $\pi$ to 5 decimal places:

**printf("pi = %.5f", 4 * atan(1.0));**

NOTE

These routines call **_doprnt**, which is an implementation-dependent routine. Each uses the variable-length argument facilities of **varargs**(3). Although it is possible to use **_doprnt** to take a list of arguments and pass them on to a routine like **printf**, not all implementations have such a routine. We strongly recommend that you use the routines described in **vprintf**(3S) instead.

SEE ALSO

**econvert**(3), **printf**(3S), **putc**(3S), **scanf**(3V), **varargs**(3), **vprintf**(3S)

BUGS

Very wide fields (>128 characters) fail.

NAME

rand, srand – simple random number generator

SYNOPSIS

**srand(seed)**
**int seed;**

**rand()**

DESCRIPTION

**rand()** uses a multiplicative congruential random number generator with period $2^{32}$ to return successive pseudo-random numbers in the range from 0 to $2^{15}-1$.

**srand()** can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NOTE

The spectral properties of **rand()** leave a great deal to be desired. **drand48**(3) and **random**(3) provide much better, though more elaborate, random-number generators.

SEE ALSO

**drand48**(3), **random**(3), **rand**(3C)

BUGS

The low bits of the numbers generated are not very random; use the middle bits. In particular the lowest bit alternates between 0 and 1.

NAME

scanf, fscanf, sscanf – formatted input conversion

SYNOPSIS

**#include <stdio.h>**

**scanf(format [ , pointer ] ... )**
**char *format;**

**fscanf(stream, format [ , pointer ] ... )**
**FILE *stream;**
**char *format;**

**sscanf(s, format [ , pointer ] ... )**
**char *s, *format;**

DESCRIPTION

**scanf( )** reads from the standard input stream **stdin**. **fscanf( )** reads from the named input stream. **sscanf( )** reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format*, described below, and a set of *pointer* arguments indicating where the converted input should be stored. The results are undefined in there are insufficient *args* for the format. If the format is exhausted while *args* remain, the excess *args* are simply ignored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (SPACE, TAB, NEWLINE, or FORMFEED) which, except in two cases described below, cause input to be read up to the next non-white-space character.
2. An ordinary character (not '%'), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character '%', an optional assignment suppressing character '*', an optional numerical maximum field width, an optional **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by '*'. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except "[" and "c", white space leading an input field is ignored.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument is given. The following conversion characters are legal:

**%** A single % is expected in the input at this point; no assignment is done.

**d** A decimal integer is expected; the corresponding argument should be an integer pointer.

**u** An unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.

**o** An octal integer is expected; the corresponding argument should be an integer pointer.

**x** A hexadecimal integer is expected; the corresponding argument should be an integer pointer.

**i** An integer is expected; the corresponding argument should be an integer pointer. It will store the value of the next input item interpreted according to C conventions: a leading "0" implies octal; a leading "0x" implies hexadecimal; otherwise, decimal.

**n** Stores in an integer argument the total number of characters (including white space) that have been scanned so far since the function call. No input is consumed.

**e,f,g** A floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is as described for **string_to_decimal**(3), with

*fortran_exponent* zero.

**s** A character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating **\0**, which will be added automatically. The input field is terminated by a white space character.

**c** A character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next non-space character, use **%1s**. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

**[** Indicates string data; the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters, which we will call the *scanset*, and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the scanset. The circumflex (^), when it appears as the first character in the scanset, serves as a complement operator and redefines the scanset as the set of all characters *not* contained in the remainder of the scanset string. There are some conventions used in the construction of the scanset. A range of characters may be represented by the construct *first–last*, thus [0123456789] may be expressed [0–9]. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the scanset. To include the right square bracket as an element of the scanset, it must appear as the first character (possibly preceded by a circumflex) of the scanset, and in this case it will not be syntactically interpreted as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating \0, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters **d**, **u**, **o**, **x**, and **i** may be preceded by **l** or **h** to indicate that a pointer to **long** or to **short** rather than to **int** is in the argument list. Similarly, the conversion characters **e**, **f**, and **g** may be preceded by **l** to indicate that a pointer to **double** rather than to **float** is in the argument list. The **l** or **h** modifier is ignored for other conversion characters.

*Avoid this common error:* because **printf**(3V) does not require that the lengths of conversion descriptors and actual parameters match, coders sometimes are careless with the **scanf( )** functions. But converting %f to &double or %lf to &float *does not work*; the results are quite incorrect.

**scanf( )** conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

**scanf( )** returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. The constant EOF is returned upon end of input. Note: this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

If the input ends before the first conflict or conversion, EOF is returned. If the input ends after the first conflict or conversion, the number of successfully matched items is returned.

## EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 thompson
```

will assign to *n* the value **3**, to *i* the value **25**, to *x* the value **5.432**, and *name* will contain **thompson\0**. Or:

```
int i, j; float x; char name[50];
(void) scanf("%i%2d%f%*d %[0-9]", &j, &i, &x, name);
```

with input:

**011 56789 0123 56a72**

will assign **9** to *j*, **56** to *i*, **789.0** to *x*, skip **0123**, and place the string **56\0** in *name*. The next call to **getchar( )** (see **getc**(3S)) will return **a**. Or:

```
int i, j, s, e; char name[50];
(void) scanf("%i %i %n%s%n", &i, &j, &s, name, &e);
```

with input:

**0x11 0xy johnson**

will assign **17** to *i*, **0** to *j*, **6** to *s*, will place the string **xy\0** in *name*, and will assign **8** to *e*. Thus, the length of *name* is $e - s = 2$. The next call to **getchar( )** (see **getc**(3S)) will return a SPACE.

## SEE ALSO

**getc**(3S), **printf**(3V), **stdio**(3V), **string_to_decimal**(3), **strtol**(3), **scanf**(3S)

## DIAGNOSTICS

These functions return EOF on end of input, and a short count for missing or illegal data items.

## BUGS

The success of literal matches and suppressed assignments is not directly determinable.

## CAVEATS

Trailing white space (including a NEWLINE) is left unread unless matched in the control string.

## NAME

setbuf, setbuffer, setlinebuf, setvbuf – assign buffering to a stream

## SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;

setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

setlinebuf(stream)
FILE *stream;

int setvbuf(stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;
```

## DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a NEWLINE is encountered or input is read from any line buffered input stream. **fflush()** (see **fclose**(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from **malloc**(3) upon the first **getc( )** or **putc**(3S) on the file.

By default, output to a terminal is line buffered, except for output to the standard stream **stderr** which is unbuffered, and all other input/output is fully buffered.

**setbuf( )** can be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered. A manifest constant **BUFSIZ**, defined in the **<stdio.h>** header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

**setbuffer**, an alternate form of **setbuf**, can be used after a stream has been opened but before it is read or written. It uses the character array *buf* whose size is determined by the *size* argument instead of an automatically allocated buffer. If *buf* is the NULL pointer, input/output will be completely unbuffered.

**setvbuf( )** can be used after a stream has been opened but before it is read or written. *type* determines how stream will be buffered. Legal values for *type* (defined in **<stdio.h>**) are:

| | |
|---|---|
| **_IOFBF** | fully buffers the input/output. |
| **_IOLBF** | line buffers the output; the buffer will be flushed when a NEWLINE is written, the buffer is full, or input is requested. |
| **_IONBF** | completely unbuffers the input/output. |

If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *size* specifies the size of the buffer to be used.

**setlinebuf( )** is used to change the buffering on a stream from block buffered or unbuffered to line buffered. Unlike **setbuf**, **setbuffer**, and **setvbuf**, it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using **freopen( )** (see **fopen**(3S)). A file can be changed from block buffered or line buffered to unbuffered by using **freopen( )** followed by **setbuf( )** with a buffer argument of NULL.

**NOTE**

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

**SEE ALSO**

**fclose**(3S), **fopen**(3V), **fread**(3S), **getc**(3S), **malloc**(3), **printf**(3V), **putc**(3S), **puts**(3S), **setbuf**(3S)

**DIAGNOSTICS**

If an illegal value for *type* or *size* is provided, **setvbuf()** returns a non-zero value. Otherwise, the value returned will be zero.

NAME

setjmp, longjmp, sigsetjmp, siglongjmp – non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;
int val;

int _setjmp(env)
jmp_buf env;

_longjmp(env, val)
jmp_buf env;
int val;

int sigsetjmp(env, savemask)
sigjmp_buf env;
int savemask;

siglongjmp(env, val)
sigjmp_buf env;
int val;
```

DESCRIPTION

**setjmp**( ) and **longjmp**( ) are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

**setjmp**( ) saves its stack environment in *env* for later use by **longjmp**. A normal call to **setjmp**( ) returns zero. **setjmp**( ) also saves the register environment. If a **longjmp**( ) call will be made, the routine which called **setjmp**( ) should not return until after the **longjmp**( ) has returned control (see below).

**longjmp**( ) restores the environment saved by the last call of **setjmp**, and then returns in such a way that execution continues as if the call of **setjmp**( ) had just returned the value *val* to the function that invoked **setjmp**; however, if *val* were zero, execution would continue as if the call of **setjmp**( ) had returned one. This ensures that a "return" from **setjmp**( ) caused by a call to **longjmp**( ) can be distinguished from a regular return from **setjmp**. The calling function must not itself have returned in the interim, otherwise **longjmp**( ) will be returning control to a possibly non-existent environment. All memory-bound data have values as of the time **longjmp**( ) was called. The CPU and floating-point data registers are restored to the values they had at the time that **setjmp**( ) was called. But, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp**. This is especially a problem for programmers trying to write machine-independent C routines.

**setjmp**( ) and **longjmp**( ) manipulate only the C stack and registers; they do not save or restore the signal mask. **_setjmp** behaves identically to **setjmp**, and **_longjmp** behaves identically to **longjmp**. If the *savemask* flag to **sigsetjmp** is non-zero, the signal mask (see **sigsetmask**(2)) is saved, and a subsequent **siglongjmp** using the same *env* will restore the signal mask. If the *savemask* flag is zero, the signal mask is not saved, and a subsequent **siglongjmp** using the same *env* will not restore the signal mask. In all other ways, **sigsetjmp** functions in the same way that **setjmp**( ) does, and **siglongjmp** functions in the same way that **longjmp**( ) does.

None of these functions save or restore any floating-point status or control registers, in particular the MC68881 **fpsr**, **fpcr**, or **fpiar**, the Sun-3 FPA **fpamode** or **fpastatus**, and the Sun-4 **%fsr**. See **ieee_flags**(3M) to save and restore floating-point status or control information.

**EXAMPLE**

The following code fragment indicates the flow of control of the **setjmp( )** and **longjmp( )** combination:

```
function declaration
...
        jmp_buf   my_environment;
          ...
        if (setjmp (my_environment)) {
          /* register variables have unpredictable values
          code after the return from longjmp
                  ...
        } else {
          /* do not modify register vars
          this is the return from setjmp
                  ...
        }
```

**SEE ALSO**

**cc**(1V), **sigsetmask**(2), **sigvec**(2), **ieee_flags**(3M), **signal**(3V), **setjmp**(3)

**BUGS**

**setjmp( )** does not save the current notion of whether the process is executing on the signal stack. The result is that a **longjmp( )** to some place on the signal stack leaves the signal stack state incorrect.

On Sun-2 and Sun-3 systems **setjmp( )** also saves the register environment. Therefore, all data that are bound to registers are restored to the values they had at the time that **setjmp( )** was called. All memory-bound data have values as of the time **longjmp( )** was called. However, because the **register** storage class is only a hint to the C compiler, variables declared as **register** variables may not necessarily be assigned to machine registers, so their values are unpredictable after a **longjmp**. When using compiler options that specify automatic register allocation (see **cc**(1V)), the compiler will not attempt to assign variables to registers in routines that call **setjmp**.

**longjmp( )** never causes **setjmp( )** to return zero in the Sun implementation; this is also true of many other implementations, including all System V implementations, so programmers should not depend on **longjmp( )** being able to cause **setjmp( )** to return zero.

**NAME**

setuid, setgid – set user and group IDs

**SYNOPSIS**

**setuid(uid)**
**setgid(gid)**

**DESCRIPTION**

**setuid( ) (setgid)** is used to set the real user (group) ID and effective user (group) ID of the calling process.

If the effective user ID of the calling process is super-user, the real user (group) ID and effective user (group) ID are set to *uid* (*gid*).

If the effective user ID of the calling process is not super-user, but its real user (group) ID is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

If the effective user (group) ID of the calling process is not super-user, but the saved set-user (group) ID from **execve**(2) is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

**SEE ALSO**

**execve**(2), **getgid**(2), **getuid**(2), **setregid**(2), **setreuid**(2),

**DIAGNOSTICS**

Zero is returned if the user (group) ID is set; −1 is returned otherwise, with the global variable **errno** set as for **setreuid( ) (setregid)**.

NAME

signal – simplified software signal facilities

SYNOPSIS

**#include <signal.h>**

**void (*signal(sig, func))()**
**void (*func)();**

DESCRIPTION

**signal()** is a simplified interface to the more general **sigvec**(2) facility. Programs that use **signal()** in preference to **sigvec()** are more likely to be portable to all systems.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see **termio**(4)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the **SIGKILL** and **SIGSTOP** signals, the **signal()** call allows signals either to be ignored or to interrupt to a specified location. The following is a list of all signals with names as in the include file **<signal.h>**:

| | | |
|---|---|---|
| **SIGHUP** | 1 | hangup |
| **SIGINT** | 2 | interrupt |
| **SIGQUIT** | 3* | quit |
| **SIGILL** | 4* | illegal instruction |
| **SIGTRAP** | 5* | trace trap |
| **SIGABRT** | 6* | abort (generated by **abort**(3) routine) |
| **SIGEMT** | 7* | emulator trap |
| **SIGFPE** | 8* | arithmetic exception |
| **SIGKILL** | 9 | kill (cannot be caught, blocked, or ignored) |
| **SIGBUS** | 10* | bus error |
| **SIGSEGV** | 11* | segmentation violation |
| **SIGSYS** | 12* | bad argument to system call |
| **SIGPIPE** | 13 | write on a pipe or other socket with no one to read it |
| **SIGALRM** | 14 | alarm clock |
| **SIGTERM** | 15 | software termination signal |
| **SIGURG** | 16• | urgent condition present on socket |
| **SIGSTOP** | 17† | stop (cannot be caught, blocked, or ignored) |
| **SIGTSTP** | 18† | stop signal generated from keyboard |
| **SIGCONT** | 19• | continue after stop (cannot be blocked) |
| **SIGCHLD** | 20• | child status has changed |
| **SIGTTIN** | 21† | background read attempted from control terminal |
| **SIGTTOU** | 22† | background write attempted to control terminal |
| **SIGIO** | 23• | I/O is possible on a descriptor (see **fcntl**(2V)) |
| **SIGXCPU** | 24 | cpu time limit exceeded (see **getrlimit**(2)) |
| **SIGXFSZ** | 25 | file size limit exceeded (see **getrlimit**(2)) |
| **SIGVTALRM** | 26 | virtual time alarm (see **getitimer**(2)) |
| **SIGPROF** | 27 | profiling timer alarm (see **getitimer**(2)) |
| **SIGWINCH** | 28• | window changed (see **termio**(4) and **win**(4S)) |
| **SIGLOST** | 29* | resource lost (see **lockd**(8C)) |
| **SIGUSR1** | 30 | user-defined signal 1 |
| **SIGUSR2** | 31 | user-defined signal 2 |

The starred signals in the list above cause a core image if not caught or ignored.

If *func* is **SIG_DFL**, the default action for signal *sig* is reinstated; this default is termination (with a core image for starred signals) except for signals marked with • or †. Signals marked with • are discarded if the action is **SIG_DFL**; signals marked with † cause the process to stop. If *func* is **SIG_IGN** the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs *func* is called. The value of *func* for the caught signal is reset to **SIG_DFL** before *func* is called, unless the signal is **SIGILL** or **SIGTRAP**

A return from the function continues the process at the point it was interrupted.

If a caught signal occurs during certain system calls, causing the call to terminate prematurely, the call is interrupted. In particular this can occur during a **read**(2V) or **write**(2V) on a slow device (such as a terminal; but not a file) and during a **wait**(2). After the signal catching function returns, the interrupted system call may return a −1 to the calling process with **errno** set to EINTR.

The value of **signal**( ) is the previous (or initial) value of *func* for the particular signal.

After a **fork**(2) or **vfork**(2) the child inherits all signals. An **execve**(2) resets all caught signals to the default action; ignored signals remain ignored.

## NOTES

The handler routine can be declared:

```
void handler(sig, code, scp, addr)
int sig, code;
struct sigcontext *scp;
char *addr;
```

Here *sig* is the signal number; *code* is a parameter of certain signals that provides additional detail; *scp* is a pointer to the **sigcontext** structure (defined in **<signal.h>**), used to restore the context from before the signal; and *addr* is additional address information. See **sigvec**(2) for more details.

## RETURN VALUE

The previous action is returned on a successful call. Otherwise, −1 is returned and **errno** is set to indicate the error.

## ERRORS

**signal**( ) will fail and no action will take place if one of the following occur:

| | |
|---|---|
| EINVAL | *sig* is not a valid signal number. |
| EINVAL | An attempt is made to ignore or supply a handler for **SIGKILL** or **SIGSTOP**. |
| EINVAL | An attempt is made to ignore **SIGCONT** (by default **SIGCONT** is ignored). |

## SEE ALSO

**kill**(1), **execve**(2), **fork**(2), **getitimer**(2), **getrlimit**(2), **kill**(2V), **ptrace**(2), **read**(2V), **sigblock**(2), **sigpause**(2), **sigsetmask**(2), **sigstack**(2), **sigvec**(2), **vfork**(2), **wait**(2), **write**(2V), **setjmp**(3), **termio**(4)

**NAME**

sleep – suspend execution for interval

**SYNOPSIS**

**unsigned sleep(seconds)**
**unsigned seconds;**

**DESCRIPTION**

**sleep()** suspends the current process from execution for the number of seconds specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) because scheduled wake-ups occur at fixed 1-second intervals and (2) because any caught signal will terminate the **sleep()** following execution of that signal's catching routine. Also, the suspension time may be an arbitrary amount longer than requested because of other activity in the system. The value returned by **sleep()** will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested **sleep()** time, or premature arousal due to another caught signal.

**sleep()** is implemented by setting an interval timer and pausing until it expires. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous value of the timer, the process sleeps only until the timer would have expired, and the signal which occurs with the expiration of the timer is sent one second later.

**SEE ALSO**

**setitimer**(2), **sigpause**(2), **usleep**(3)

## NAME

stdio – standard buffered input/output package

## SYNOPSIS

**#include <stdio.h>**

**FILE *stdin;**
**FILE *stdout;**
**FILE *stderr;**

## DESCRIPTION

The functions described in sections 3V and 3S constitute a user-level I/O buffering scheme. The in-line macros **getc**(3V) and **putc**(3S) handle characters quickly. The macros **getchar** and **putchar**, and the higher level routines **fgetc, getw, gets, fgets, scanf, fscanf, fread, fputc, putw, puts, fputs, printf, fprintf, fwrite** all use or act as if they use **getc( )** and **putc( )** ; they can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type **FILE**. **fopen**(3V) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the **<stdio.h>** include file and associated with the standard open files:

| | |
|---|---|
| **stdin** | standard input file |
| **stdout** | standard output file |
| **stderr** | standard error file |

A constant **NULL** (0) designates a nonexistent pointer.

An integer constant **EOF** (–1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

Any module that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in sections labeled 3V and 3S of this manual are declared in that header file and need no further declaration. The constants and the following 'functions' are implemented as macros; redeclaration of these names is perilous: **getc, getchar, putc, putchar, feof, ferror, fileno,** and **clearerr**.

Output streams, with the exception of the standard error stream **stderr**, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream **stderr** is by default unbuffered, but use of **fopen**(3V) will cause it to become buffered or line-buffered. When an output stream is unbuffered, information is written to the destination file or terminal as soon as it is output to the stream; when it is buffered, many characters are saved up and written as a block. When it is line-buffered, each line of output is written to the destination file or terminal as soon as the line is completed (that is, as soon as a NEWLINE character is output or as soon as input is read from a line-buffered stream). **setbuf**(3V), **setbuffer, setlinebuf,** or **setvbuf** can be used to change the stream's buffering strategy.

## SEE ALSO

**open**(2V), **close**(2), **lseek**(2), **pipe**(2), **read**(2V), **vfork**(2), **write**(2V), **ctermid**(3S), **cuserid**(3S), **fclose**(3S), **ferror**(3V), **fopen**(3V), **fread**(3S), **fseek**(3S), **getc**(3V), **gets**(3S), **popen**(3S), **printf**(3V), **putc**(3S), **puts**(3S), **scanf**(3V), **setbuf**(3V), **system**(3), **tmpfile**(3S), **tmpnam**(3S), **ungetc**(3S)

## DIAGNOSTICS

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with **fopen**, input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

## BUGS

The standard buffered functions do not interact well with certain other library and system functions, especially **vfork**(2).

**NOTES**

The line buffering of output to terminals is almost always transparent, but may cause confusion or malfunctioning of programs which use standard I/O routines but use **read**(2V) to read from the standard input, as calls to **read**( ) do not cause output to line-buffered streams to be flushed.

Output saved up on *all* line-buffered streams is written when input is read from *any* line-buffered stream. Input read from a stream that is not line-buffered does not flush output on line-buffered streams.

In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to call **fflush** (see **fclose**(3S)) on the standard output before performing the computation so that the output will appear.

## NAME

times – get process and child process times

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/times.h>

long times(buffer)
struct tms *buffer;
```

## DESCRIPTION

**times**( ) returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ is 60.

This is the structure returned by **times**:

```
struct tms {
        time_t  tms_utime;          /* user time */
        time_t  tms_stime;          /* system time */
        time_t  tms_cutime;         /* user time, children */
        time_t  tms_cstime;         /* system time, children */
};
```

This information comes from the calling process and each of its terminated child processes for which it has executed a **wait**.

**tms_utime** is the CPU time used while executing instructions in the user space of the calling process.

**tms_stime** is the CPU time used by the system on behalf of the calling process.

**tms_cutime** is the sum of the **tms_utimes** and **tms_cutimes** of the child processes.

**tms_cstime** is the sum of the **tms_stimes** and **tms_cstimes** of the child processes.

## RETURN VALUE

Upon successful completion, **times**( ) returns the elapsed real time, in 60ths of a second, since an arbitrary point in the past. This point does not change from one invocation of **times**( ) to another within the same process. If **times**( ) fails, a −1 is returned and **errno** is set to indicate the error.

## SEE ALSO

**time**(1V), **getrusage**(2), **wait**(2), **time**(3C)

**NAME**

ttyslot – find the slot in the utmp file of the current process

**SYNOPSIS**

**ttyslot()**

**DESCRIPTION**

**ttyslot()** returns the index of the current user's entry in the **/etc/utmp** file. This is accomplished by actually scanning the file **/etc/ttys** for the name of the terminal associated with the standard input, the standard output, or the error output (0, 1 or 2).

**FILES**

**/etc/ttys**
**/etc/utmp**

**DIAGNOSTICS**

A value of −1 is returned if an error was encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

NAME

vprintf, vfprintf, vsprintf – print formatted output of a varargs argument list

SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf(format, ap)
char *format;
va_list ap;

int vfprintf(stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf(s, format, ap)
char *s, *format;
va_list ap;
```

DESCRIPTION

**vprintf**, **vfprintf**, and **vsprintf**( ) are the same as **printf**(3V), **fprintf**, and **sprintf** respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by **varargs**(3).

EXAMPLE

The following demonstrates how **vfprintf**( ) could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
...
        /* error should be called like:
         *      error(function_name, format, arg1, arg2...);
         * Note that function_name and format cannot be declared
         * separately because of the definition of varargs.
         */

/*VARARGS0*/
void
error (va_alist)
        va_dcl;
{
        va_list args;
        char *fmt;

        va_start(args);
                /* print name of function causing error */
        (void) fprintf(stderr, "ERROR in %s: ", va_arg(args, char *));
        fmt = va_arg(args, char *);
                /* print out remainder of message */
        (void) vfprintf(stderr, fmt, args);
        va_end(args);
        (void) abort( );
}
```

SEE ALSO

**printf**(3V), **varargs**(3)

## NAME

intro – introduction to device drivers, protocols, and network interfaces

## DESCRIPTION

This section describes device drivers, high-speed network interfaces, and protocols available under SunOS. The system provides drivers for a variety of hardware devices, such as disks, magnetic tapes, serial communication lines, mice and frame buffers, as well as virtual devices such as pseudo-terminals and windows. SunOS provides hardware support and a network interface for the 10-Megabit Ethernet, along with interfaces for the IP protocol family and a STREAMS-based Network Interface Tap (NIT) facility.

In addition to describing device drivers that are supported by the 4.3BSD operating system, this section contains subsections that describe:

- SunOS-specific device drivers, under '4S'.
- Protocol families, under '4F'.
- Protocols and raw interfaces, under '4P'.
- STREAMS modules, under '4M'.
- Network interfaces, under '4N'.

### Configuration

The SunOS kernel can be configured to include or omit many of the device drivers described in this section. The CONFIG section of the manual page gives the line(s) to include in the kernel configuration file for each machine architecture on which a device is supported. If no specific architectures are indicated, the configuration syntax applies to all Sun systems.

The GENERIC kernel is the default configuration for SunOS. It contains all of the optional drivers for a given machine architecture. See **config**(8), for details on configuring a new SunOS kernel.

The manual page for a device driver may also include a DIAGNOSTICS section, listing error messages that the driver might produce. Normally, these messages are logged to the appropriate system log using the kernel's standard message-buffering mechanism (see **syslogd**(8)); they may also appear on the system console.

### Ioctls

Various special functions, such as querying or altering the operating characteristics of a device, are performed by supplying appropriate parameters to the **ioctl**(2) system call. These parameters are often referred to as "ioctls." Ioctls for a specific device are presented in the manual page for that device. Ioctls that pertain to a class of devices are listed in a manual page with a name that suggests the class of device, and ending in 'io', such as **mtio**(4) for magnetic tape devices, or **dkio**(4S) for disk controllers. In addition, some ioctls operate directly on higher-level objects such as files, terminals, sockets, and streams:

- Ioctls that operate directly on files, file descriptors, and sockets are described in **filio**(4). Note: the **fcntl**(2) system call is the primary method for operating on file descriptors as such, rather than on the underlying files. Also note that the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. Ioctls for a specific network interface are documented in the manual page for that interface.
- Ioctls for terminals, including pseudo-terminals, are described in **termio**(4). This manual page includes information about both the BSD **termios** structure, as well as the System V **termio** structure.
- Ioctls for STREAMS are described in **streamio**(4).

### Devices Always Present

Device drivers present in every kernel include:

- The paging device; see **drum**(4).
- Drivers for accessing physical, virtual, and I/O space in memory; see **mem**(4S).
- The data sink; see **null**(4).

**Terminals and Serial Communications Devices**

Serial communication lines are normally supported by the terminal driver; see **tty**(4). This driver manages serial lines provided by communications drivers, such as those described in **mti**(4S) and **zs**(4S). The terminal driver also handles serial lines provided by virtual terminals, such as the Sun console monitor described in **console**(4S), and true pseudo-terminals, described in **pty**(4).

**Disk Devices**

Drivers for the following disk controllers provide standard block and raw interfaces under SunOS;

- SCSI controllers, in **sd**(4S),
- Xylogics 450 and 451 SMD controllers, in **xy**(4S),
- Xylogics 7053 SMD controllers, in **xd**(4S).

Ioctls to query or set a disk's geometry and partitioning are described in **dkio**(4S).

**Magnetic Tape Devices**

Magnetic tape devices supported by SunOS include those described in **ar**(4S), **tm**(4S), **st**(4S), and **xt**(4S). Ioctls for all tape-device drivers are described in **mtio**(4S).

**Frame Buffers**

Frame buffer devices include color frame buffers described in the **cg***(4S) manual pages, monochrome frame buffers described in the **bw***(4S) manual pages, graphics processor interfaces described in the **gp***(4S) manual pages, and an indirect device for the console frame buffer described in **fb**(4S). Ioctls for all frame-buffer devices are described in **fbio**(4S).

**Miscellaneous Devices**

Miscellaneous devices include the console keyboard described in **kbd**(4S), the console mouse described in **mouse**(4S), window devices described in **win**(4S), and the DES encryption-chip interface described in **des**(4S).

**Network-Interface Devices**

SunOS supports the 10-Megabit Ethernet as its primary network interface; see **ec**(4S), **ie**(4S), and **le**(4S) for details. However, a software loopback interface, **lo**(4) is also supported. General properties of these network interfaces are described in **if**(4N), along with the ioctls that operate on them.

Support for network routing is described in **routing**(4N).

**Protocols and Protocol Families**

SunOS supports both socket-based and STREAMS-based network communications. The Internet protocol family, described in **inet**(4F), is the primary protocol family primary supported by SunOS, although the system can support a number of others. The raw interface provides low-level services, such as packet fragmentation and reassembly, routing, addressing, and basic transport for socket-based implementations. Facilities for communicating using an Internet-family protocol are generally accessed by specifying the **AF_INET** address family when binding a socket; see **socket**(2) for details.

Major protocols in the Internet family include:

- The Internet Protocol (IP) itself, which supports the universal datagram format, as described in **ip**(4P). This is the default protocol for SOCK_RAW type sockets within the **AF_INET** domain.
- The Transmission Control Protocol (TCP); see **tcp**(4P). This is the default protocol for SOCK_STREAM type sockets.
- The User Datagram Protocol (UDP); see **udp**(4P). This is the default protocol for SOCK_DGRAM type sockets.
- The Address Resolution Protocol (ARP); see **arp**(4P).
- The Internet Control Message Protocol (ICMP); see **icmp**(4P).

The Network Interface Tap (NIT) protocol, described in **nit**(4P), is a STREAMS-based facility for accessing the network at the link level.

SEE ALSO

**fcntl**(2), **getsockopt**(2), **ioctl**(2), **socket**(2), **ar**(4S), **arp**(4P), **dkio**(4S), **drum**(4), **ec**(4S), **fb**(4S), **fbio**(4S), **filio**(4), **icmp**(4P), **if**(4N), **inet**(4F), **ip**(4P), **kbd**(4S), **le**(4), **lo**(4), **mbio**(4S), **mem**(4S), **mti**(4), **mtio**(4), **nit**(4P), **null**(4), **pty**(4), **routing**(4N), **sd**(4S), **st**(4S) **streamio**(4), **tcp**(4P), **termio**(4), **tm**(4S), **tty**(4), **udp**(4P), **win**(4S), **xd**(4S), **xy**(4S), **zs**(4S)

LIST OF DEVICES, INTERFACES AND PROTOCOLS

| Name | Appears on Page | Description |
|---|---|---|
| **alm** | **mcp**(4S) | Asynchronous Line Multiplexer |
| **ar** | **ar**(4S) | Archive 1/4 inch Streaming Tape Drive |
| **arp** | **arp**(4P) | Address Resolution Protocol |
| **bk** | **bk**(4) | line discipline for machine-machine communication |
| **bwone** | **bwone**(4S) | Sun-1 black and white frame buffer |
| **bwtwo** | **bwtwo**(4S) | Sun-3/Sun-2 black and white frame buffer |
| **cgfour** | **cgfour**(4S) | Sun-3 color memory frame buffer |
| **cgone** | **cgone**(4S) | Sun-1 color graphics interface |
| **cgthree** | **cgthree**(4S) | Sun386i color memory frame buffer |
| **cgtwo** | **cgtwo**(4S) | Sun-3/Sun-2 color graphics interface |
| **clone** | **clone**(4) | open any minor device on a STREAMS driver |
| **console** | **console**(4S) | console driver and terminal emulator for the Sun workstation |
| **des** | **des**(4S) | DES encryption chip interface |
| **dkio** | **dkio**(4S) | generic disk control operations |
| **drum** | **drum**(4) | paging device |
| **ec** | **ec**(4S) | 3Com 10 Mb/s Ethernet interface |
| **fb** | **fb**(4S) | driver for Sun console frame buffer |
| **fbio** | **fbio**(4S) | general properties of frame buffers |
| **fd** | **fd**(4S) | Disk driver for Floppy Disk Controllers |
| **filio** | **filio**(4) | ioctls that operate directly on files, file descriptors, and sockets |
| **fpa** | **fpa**(4S) | Sun-3 floating point accelerator |
| **gpone** | **gpone**(4S) | Sun-3/Sun-2 graphics processor |
| **icmp** | **icmp**(4P) | Internet Control Message Protocol |
| **ie** | **ie**(4S) | Intel 10 Mb/s Ethernet interface |
| **if** | **if**(4N) | general properties of network interfaces |
| **inet** | **inet**(4F) | Internet protocol family |
| **ip** | **ip**(4P) | Internet Protocol |
| **kb** | **kb**(4M) | Sun keyboard STREAMS module |
| **kbd** | **kbd**(4S) | Sun keyboard |
| **kmem** | **mem**(4S) | main memory and bus I/O space |
| **ldterm** | **ldterm**(4M) | standard terminal STREAMS module |
| **le** | **le**(4S) | Sun-3/50, Sun-3/60 10MB Ethernet interface |
| **lo** | **lo**(4) | software loopback network interface |
| **lofs** | **lofs**(4S) | loopback virtual file system |
| **mbio** | **mem**(4S) | main memory and bus I/O space |
| **mbmem** | **mem**(4S) | main memory and bus I/O space |
| **mcp** | **mcp**(4S) | MCP Multiprotocol Communications Processor |
| **mem** | **mem**(4S) | main memory and bus I/O space |
| **mouse** | **mouse**(4S) | Sun mouse |
| **ms3** | **mouse**(4S) | Sun mouse |
| **ms** | **ms**(4M) | Sun mouse STREAMS module |
| **mti** | **mti**(4S) | Systech MTI-800/1600 multi-terminal interface |
| **mtio** | **mtio**(4) | UNIX system magnetic tape interface |

| | | |
|---|---|---|
| **NFS** | **nfs**(4P) | network file system |
| **nif_pf** | **nit_pf**(4M) | streams NIT packet filtering module |
| **nit** | **nit**(4P) | Network Interface Tap facility |
| **nit_buf** | **nit_buf**(4M) | streams NIT buffering module |
| **nit_if** | **nit_if**(4M) | streams NIT device interface module |
| **null** | **null**(4) | data sink |
| **pp** | **pp**(4) | Centronics-compatible parallel printer port |
| **pty** | **pty**(4) | pseudo terminal driver |
| **root** | **root**(4S) | pseudo-driver for Sun root disk |
| **routing** | **routing**(4N) | system supporting for local network packet routing |
| **sd** | **sd**(4S) | Disk driver for SCSI Disk Controllers |
| **st** | **st**(4S) | Sysgen SC 4000 and Emulex MT-02 Tape Controller |
| **streamio** | **streamio**(4) | STREAMS ioctl commands |
| **tcp** | **tcp**(4P) | Transmission Control Protocol |
| **termio** | **termio**(4) | general terminal interface |
| **tm** | **tm**(4S) | tapemaster 1/2 inch tape drive |
| **ttcompat** | **ttcompat**(4M) | V7/4BSD compatibility STREAMS module |
| **tty** | **tty**(4) | controlling terminal interface |
| **udp** | **udp**(4P) | User Datagram Protocol |
| **vme16d16** | **mem**(4S) | main memory and bus I/O space |
| **vme16d32** | **mem**(4S) | main memory and bus I/O space |
| **vme24d16** | **mem**(4S) | main memory and bus I/O space |
| **vme24d32** | **mem**(4S) | main memory and bus I/O space |
| **vme32d16** | **mem**(4S) | main memory and bus I/O space |
| **vme32d32** | **mem**(4S) | main memory and bus I/O space |
| **vp** | **vp**(4S) | Ikon 10071-5 Versatec parallel printer interface |
| **vpc** | **vpc**(4S) | Systech VPC-2200 Versatec plotter and Centronics printer |
| **win** | **win**(4S) | Sun window system |
| **xd** | **xd**(4S) | Disk driver for Xylogics 7053 SMD Disk Controller |
| **xt** | **xt**(4S) | Xylogics 472 1/2 inch tape controller |
| **xy** | **xy**(4S) | Disk driver for Xylogics SMD Disk Controllers |
| **zero** | **zero**(4S) | source of zeroes |
| **zs** | **zs**(4S) | Zilog 8530 SCC serial comunications driver |

NAME

ar – Archive 1/4 inch Streaming Tape Drive

CONFIG — SUN-2 SYSTEM

**device ar0 at mbio ? csr 0x200 priority 3**
**device ar1 at mbio ? csr 0x208 priority 3**

AVAILABILITY

Sun-2, Sun-3, and Sun-4 systems only.

DESCRIPTION

The Archive tape controller is a Sun 'QIC-II' interface to an Archive streaming tape drive. It provides a standard tape interface to the device, see **mtio**(4), with some deficiencies listed under BUGS below.

The maximum blocksize for the raw device is limited only by available memory.

FILES

**/dev/rar***
**/dev/nrar*** non-rewinding

SEE ALSO

**mtio**(4)

DIAGNOSTICS

**ar*: would not initialize**

**ar*: already open**
The tape can be open by only one process at a time

**ar*: no such drive**

**ar*: no cartridge in drive**

**ar*: cartridge is write protected**

**ar: interrupt from unitialized controller %x**

**ar*: many retries, consider retiring this**

**ar*: %b error at block #**

**ar*: %b error at block #**

**ar: giving up on Rdy, try**

BUGS

The tape cannot reverse direction so the **BSF** and **BSR** ioctls are not supported.

The **FSR** ioctl is not supported.

The system will hang if the tape is removed while running.

When using the raw device, the number of bytes in any given transfer must be a multiple of 512 bytes. If it is not, the device driver returns an error.

The driver will only write an EOF mark on close if the last operation was a write, without regard for the mode used when opening the file. This delete empty files on a raw tape copy operation.

NAME

arp – Address Resolution Protocol

CONFIG

**pseudo-device ether**

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if_arp.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
```

DESCRIPTION

ARP is a protocol used to dynamically map between DARPA Internet Protocol (IP) and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet interface drivers. It is not specific to the Internet Protocol or to the 10Mb/s Ethernet, but this implementation currently supports only that combination.

ARP caches IP-to-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently "transmitted" packet is kept.

To facilitate communications with systems which do not use ARP, **ioctl**s are provided to enter and delete entries in the IP-to-Ethernet tables.

USAGE

```
#include <sys/sockio.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.h>
struct arpreq arpreq;
ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each ioctl takes the same structure as an argument. **SIOCSARP** sets an ARP entry, **SIOCGARP** gets an ARP entry, and **SIOCDARP** deletes an ARP entry. These ioctls may be applied to any socket descriptor *s*, but only by the super-user. The **arpreq** structure contains:

```
/*
 * ARP ioctl request
 */
struct arpreq {
        struct sockaddr  arp_pa;        /* protocol address */
        struct sockaddr  arp_ha;        /* hardware address */
        int      arp_flags;             /* flags */
};
/*  arp_flags field values */
#define ATF_COM                 0x2     /* completed entry (arp_ha valid) */
#define  ATF_PERM               0x4     /* permanent entry */
#define  ATF_PUBL               0x8     /* publish (respond for other host) */
#define  ATF_USETRAILERS                0x10    /* send trailer packets to host */
```

The address family for the **arp_pa** sockaddr must be **AF_INET**; for the **arp_ha** sockaddr it must be **AF_UNSPEC**. The only flag bits which may be written are **ATF_PERM**, **ATF_PUBL** and **ATF_USETRAILERS**. **ATF_PERM** makes the entry permanent if the **ioctl** call succeeds. The peculiar nature of the ARP tables may cause the **ioctl** to fail if more than 6 (permanent) IP addresses hash to the same slot. **ATF_PUBL** specifies that the ARP code should respond to ARP requests for the indicated host

coming from other machines. This allows a host to act as an "ARP server" which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts which wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The **ATF_USETRAILERS** flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (that is, a host which responds to an ARP mapping request for the local host's address).

## SEE ALSO

**ec**(4S), **ie**(4S), **inet**(4F), **arp**(8C), **ifconfig**(8C)

Plummer, Dave, "*An Ethernet Address Resolution Protocol -or- Converting Network Protocol Addresses to 48.bit Ethernet Addresses for Transmission on Ethernet Hardware*," RFC 826, Network Information Center, SRI International, Menlo Park, Calif., November 1982. (Sun 800-1059-10)

Leffler, Sam, and Michael Karels, "*Trailer Encapsulations*," RFC 893, Network Information Center, SRI International, Menlo Park, Calif., April 1984.

## DIAGNOSTICS

**duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x.** ARP has discovered another host on the local network which responds to mapping requests for its own Internet address.

## BUGS

ARP packets on the Ethernet use only 42 bytes of data, however, the smallest legal Ethernet packet is 60 bytes (not including CRC). Some systems may not enforce the minimum packet size, others will.

## NAME

bk – line discipline for machine-machine communication

## SYNOPSIS

**pseudo-device bk**

## DESCRIPTION

This line discipline provides a replacement for the tty driver **tty**(4) when high speed output to and especially input from another machine is to be transmitted over an asynchronous communications line. The discipline was designed for use by a (now obsolete) store-and-forward local network running over serial lines. It may be suitable for uploading of data from microprocessors into the system. If you are going to send data over asynchronous communications lines at high speed into the system, you must use this discipline, as the system otherwise may detect high input data rates on terminal lines and disable the lines; in any case the processing of such data when normal terminal mechanisms are involved saturates the system.

The line discipline is enabled by a sequence:

```
#include <sgtty.h>
int ldisc = NETLDISC, fildes; ...
ioctl(fildes, TIOCSETD, &ldisc);
```

A typical application program then reads a sequence of lines from the terminal port, checking header and sequencing information on each line and acknowledging receipt of each line to the sender, who then transmits another line of data. Typically several hundred bytes of data and a smaller amount of control information will be received on each handshake.

The old standard teletype discipline can be restored by doing:

```
ldisc = OTTYDISC;
ioctl(fildes, TIOCSETD, &ldisc);
```

While in networked mode, normal teletype output functions take place. Thus, if an 8 bit output data path is desired, it is necessary to prepare the output line by putting it into RAW mode using **ioctl**(2). This must be done *before* changing the discipline with TIOCSETD, as most **ioctl**(2) calls are disabled while in network line-discipline mode.

When in network mode, input processing is very limited to reduce overhead. Currently the input path is only 7 bits wide, with newline the only character terminating an input record. Each input record must be read and acknowledged before the next input is read as the system refuses to accept any new data when there is a record in the buffer. The buffer is limited in length, but the system guarantees to always be willing to accept input resulting in 512 data characters and then the terminating newline.

User level programs should provide sequencing and checksums on the information to guarantee accurate data transfer.

## SEE ALSO

**ioctl**(2), **tty**(4)

**NAME**

bwone – Sun-1 black and white frame buffer

**CONFIG — SUN-2 SYSTEM**

**device bwone0 at mbmem ? csr 0xc0000 priority 3**

**DESCRIPTION**

The **bwone** interface provides access to Sun-1 system black and white graphics controller boards. It supports the ioctls described in **fbio**(4S).

**FILES**

**/dev/bwone[0-9]**

**SEE ALSO**

**mmap**(2), **fb**(4S), **fbio**(4S)

**BUGS**

Use of vertical-retrace interrupts is not supported.

The video state returned by the **FBIOGVIDEO** ioctl may be incorrect. It is not possible for the driver to determine the state of the hardware video enable bit, so it reports the last state stored by the **FBIOSVIDEO** ioctl. User processes which map the frame buffer can directly enable or disable the video, unknown to the driver.

**NAME**

bwtwo – Sun-3/Sun-2 black and white frame buffer

**CONFIG — SUN-3 SYSTEM**

**device bwtwo0 at obmem 1 csr 0xff000000 priority 4**
**device bwtwo0 at obmem 2 csr 0x100000 priority 4**
**device bwtwo0 at obmem 3 csr 0xff000000 priority 4**
**device bwtwo0 at obmem 4 csr 0xff000000**
**device bwtwo0 at obmem 7 csr 0xff000000 priority 4**

The first synopsis line given above should be used to generate a kernel for a Sun-3/75, Sun-3/140 or Sun-3/160 system; the second, for a Sun-3/50 system; the third, for a Sun-3/260 system; the fourth, for a Sun-3/110 system; and the fifth, for a Sun-3/60 system.

**CONFIG — SUN-2 SYSTEM**

**device bwtwo0 at obmem 1 csr 0x700000 priority 4**
**device bwtwo0 at obio 2 csr 0x0 priority 4**

The first synopsis line given above should be used to generate a kernel for a Sun-2/120 or Sun-2/170 system; the second, for a Sun-2/50 or Sun-2/160 system.

**CONFIG — Sun386i SYSTEM**

**device bwtwo0 at obmem ? csr 0xA0200000**

**DESCRIPTION**

The **bwtwo** interface provides access to Sun monochrome memory frame buffers. It supports the ioctls described in **fbio**(4S).

If **flags 0x1** is specified, frame buffer write operations are buffered through regular high-speed RAM. This "copy memory" mode of operation speeds frame buffer accesses, but consumes an extra 128K bytes of memory. Only Sun-2, Sun-3/75, and Sun-3/160 systems support copy memory; on other systems a warning message is printed and the flag is ignored.

Reading or writing to the frame buffer is not allowed — you must use the **mmap**(2) system call to map the board into your address space.

**FILES**

**/dev/bwtwo[0-9]**

**SEE ALSO**

**mmap**(2), **cgfour**(4S), **fb**(4S), **fbio**(4S)

**BUGS**

Use of vertical-retrace interrupts is not supported.

## NAME

cgfour – Sun-3 color memory frame buffer

## CONFIG — SUN-3 SYSTEM

**device cgfour0 at obmem 4 csr 0xff000000 priority 4**
**device cgfour0 at obmem 7 csr 0xff000000 priority 4**

The first synopsis line given should be used to generate a kernel for the Sun-3/110 system; the second, for a Sun-3/60 system.

## DESCRIPTION

The **cgfour** is a color memory frame buffer with a monochrome overlay plane and an overlay enable plane implemented on the Sun-3/110 system and some Sun-3/60 system models. It provides the standard frame buffer interface as defined in **fbio**(4S).

In addition to the ioctls described under **fbio**(4S), the **cgfour** interface responds to two **cgfour**-specific colormap ioctls, **FBIOPUTCMAP** and **FBIOGETCMAP**. **FBIOPUTCMAP** returns no information other than success/failure using the ioctl return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in **<sun/fbio.h>** as:

```
struct fbcmap {
        int             index;          /* first element (0 origin) */
        int             count;          /* number of elements */
        unsigned char   *red;           /* red color map elements */
        unsigned char   *green;         /* green color map elements */
        unsigned char   *blue;          /* blue color map elements */
};
```

The driver uses color board vertical-retrace interrupts to load the colormap.

The Sun-3/60 system has an overlay plane colormap, which is accessed by encoding the plane group into the index value with the PIX_GROUP macro (see **<pixrect/pr_planegroups.h>**).

## FILES

**/dev/cgfour0**

## SEE ALSO

**mmap**(2), **fbio**(4S)

**NAME**

cgone – Sun-1 color graphics interface

**CONFIG — SUN-2 SYSTEM**

**device cgone0 at mbmem ? csr 0xec000 priority 3**

**DESCRIPTION**

The **cgone** interface provides access to the Sun-1 system color graphics controller board, which is normally supplied with a 13'' or 19'' RS170 color monitor. It provides the standard frame buffer interface as defined in **fbio**(4S).

It supports the **FBIOGPIXRECT** ioctl which allows SunView to be run on it; see **fbio**(4S)

The hardware consumes 16 kilobytes of Multibus memory space. The board starts at standard addresses 0xE8000 or 0xEC000. The board must be configured for interrupt level 3.

**FILES**

**/dev/cgone[0-9]**

**SEE ALSO**

**mmap**(2), **fbio**(4S)

**BUGS**

Use of color board vertical-retrace interrupts is not supported.

**NAME**

cgthree – Sun386i color memory frame buffer

**CONFIG**

**device cgthree0 at obmem ? csr 0xA0400000**

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

**cgthree** is a color memory frame buffer. It provides the standard frame buffer interface as defined in **fbio**(4S).

In addition to the ioctls described under **fbio**(4S), the **cgthree** interface responds to two **cgthree**-specific colormap **ioctl**(2) parameters, **FBIOPUTCMAP and .SB FBIOGETCMAP. FBIOPUTCMAP** returns no information other than success/failure via the **ioctl** return value. **FBIOGETCMAP** returns its information in the arrays pointed to by the red, green, and blue members of its **fbcmap** structure argument; **fbcmap** is defined in *<sun/fbio.h>* as:

```
struct fbcmap {
        int             index;          /* first element (0 origin) */
        int             count;          /* number of elements */
        unsigned char   *red;           /* red color map elements */
        unsigned char   *green;         /* green color map elements */
        unsigned char   *blue;          /* blue color map elements */
};
```

**FILES**

**/dev/cgthree0**

**SEE ALSO**

**mmap**(2), **fbio**(4S)

**NAME**

cgtwo – Sun-3/Sun-2 color graphics interface

**CONFIG — SUN-3 SYSTEM**

**cgtwo0 at vme24d16 ? csr 0x400000**

**CONFIG — SUN-2 SYSTEM**

**cgtwo0 at vme24 ? csr 0x400000**

**DESCRIPTION**

The **cgtwo** interface provides access to the Sun-3/Sun-2 system color graphics controller board, which is normally supplied with a 19'' 66 Hz non-interlaced color monitor. It provides the standard frame buffer interface as defined in **fbio**(4S).

The hardware consumes 4 megabytes of VME bus address space. The board starts at standard address 0x400000. The board must be configured for interrupt level 3.

**FILES**

**/dev/cgtwo[0-9]**

**SEE ALSO**

**mmap**(2), **fbio**(4S)

## NAME

clone – open any minor device on a STREAMS driver

## DESCRIPTION

**clone** is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to **clone** during the open operation is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate stream to a previously unused minor device.

The **clone** driver supports only an **open**(2V) function. This open function performs all of the necessary work so that subsequent system calls (including **close(2)**) require no further involvement of the **clone** driver.

## ERRORS

**clone** generates an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

## CAVEATS

Multiple opens of the same minor device are not supported through the **clone** interface. Executing **stat**(2) on the file system node for a cloned device yields a different result than does executing **fstat** using a file descriptor obtained from opening that node.

## SEE ALSO

**close**(2), **open**(2V), **stat**(2)

NAME

console – console driver and terminal emulator for the Sun workstation

CONFIG

None; included in standard system.

SYNOPSIS

**#include <fcntl.h>**
**#include <sys/termios.h>**
**open("/dev/console", mode);**

DESCRIPTION

**console** is an indirect driver for the Sun console terminal. On a Sun workstation, this driver refers to the workstation console driver, which implements a standard UNIX system terminal. On a Sun server without a keyboard or a frame buffer, this driver refers to the CPU serial port driver (**zs**(4S)); a terminal is normally connected to this port.

The workstation console does not support any of the **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure or by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure, as these functions apply only to asynchronous serial ports. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a slave device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

The workstation console driver calls the PROM resident monitor to output data to the console frame buffer. Keystrokes from the CPU serial port to which the keyboard is connected are routed through the keyboard STREAMS module (**kb**(4M)) and treated as input.

When the Sun window system **win**(4S) is active, console input is directed through the window system rather than being treated as input by the workstation console driver.

IOCTLS

An ioctl **TIOCCONS** can be applied to pseudo-terminals (**pty**(4)) to route output that would normally appear on the console to the pseudo-terminal instead. Thus, the window system does a **TIOCCONS** on a pseudo-terminal so that the system will route console output to the window to which that pseudo-terminal is connnected, rather than routing output through the PROM monitor to the screen, since routing output through the PROM monitor destroys the integrity of the screen. Note: when you use **TIOCCONS** in this way, the console *input* is routed from the pseudo-terminal as well.

If a **TIOCCONS** is performed on **/dev/console**, or the pseudo-terminal to which console output is being routed is closed, output to the console will again be routed to the workstation console driver.

ANSI STANDARD TERMINAL EMULATION

The Sun Workstation's PROM monitor provides routines that emulates a standard ANSI X3.64 terminal.

Note: the VT100 also follows the ANSI X3.64 standard but both the Sun and the VT100 have nonstandard extensions to the ANSI X3.64 standard. The Sun terminal emulator and the VT100 are *not* compatible in any true sense.

The Sun console displays 34 lines of 80 ASCII characters per line, with scrolling, $(x, y)$ cursor addressability, and a number of other control functions.

The Sun console displays a non-blinking block cursor which marks the current line and character position on the screen. ASCII characters between 0x20 (space) and 0x7E (tilde) inclusive are printing characters — when one is written to the Sun console (and is not part of an escape sequence), it is displayed at the current cursor position and the cursor moves one position to the right on the current line. If the cursor is already at the right edge of the screen, it moves to the first character position on the next line. If the cursor is already at the right edge of the screen on the bottom line, the Line-feed function is performed (see control–J below), which scrolls the screen up by one or more lines or wraps around, before moving the cursor to the first character position on the next line.

**Control Sequence Syntax**

The Sun console defines a number of control sequences which may occur in its input. When such a sequence is written to the Sun console, it is not displayed on the screen, but effects some control function as described below, for example, moves the cursor or sets a display mode.

Some of the control sequences consist of a single character. The notation

**control–***X*

for some character *X* , represents a control character.

Other ANSI control sequences are of the form

**ESC [** <*params*> <*char*>

Spaces are included only for readability; these characters must occur in the given sequence without the intervening spaces.

ESC represents the ASCII escape character (ESC, control–[, 0x1B).

[ The next character is a left square bracket '[' (0x5B).

<*params*>
are a sequence of zero or more decimal numbers made up of digits between 0 and 9, separated by semicolons.

<*char*> represents a function character, which is different for each control sequence.

Some examples of syntactically valid escape sequences are (again, ESC represent the single ASCII character 'Escape'):

| | |
|---|---|
| ESC[m | *select graphic rendition with default parameter* |
| ESC[7m | *select graphic rendition with reverse image* |
| ESC[33;54H | *set cursor position* |
| ESC[123;456;0;;3;B | *move cursor down* |

Syntactically valid ANSI escape sequences which are not currently interpreted by the Sun console are ignored. Control characters which are not currently interpreted by the Sun console are also ignored.

Each control function requires a specified number of parameters, as noted below. If fewer parameters are supplied, the remaining parameters default to 1, except as noted in the descriptions below.

If more than the required number of parameters is supplied, only the last *n* are used, where *n* is the number required by that particular command character. Also, parameters which are omitted or set to zero are reset to the default value of 1 (except as noted below).

Consider, for example, the command character M which requires one parameter. ESC[;M and ESC[0M and ESC[M and ESC[23;15;32;1M are all equivalent to ESC[1M and provide a parameter value of 1. Note: ESC[;5M (interpreted as 'ESC[5M') is *not* equivalent to ESC[5;M (interpreted as 'ESC[5;1M') which is ultimately interpreted as 'ESC[1M').

In the syntax descriptions below, parameters are represented as '**#**' or '**#1;#2**'.

**ANSI Control Functions**

The following paragraphs specify the ANSI control functions implemented by the Sun console. Each description gives:

- the control sequence syntax
- the hex equivalent of control characters where applicable
- the control function name and ANSI or Sun abbreviation (if any).
- description of parameters required, if any
- description of the control function
- for functions which set a mode, the initial setting of the mode. The initial settings can be restored with the SUNRESET escape sequence.

**Control Character Functions**

control–G (0x7) Bell (BEL)

The Sun Workstation Model 100 and 100U is not equipped with an audible bell. It 'rings the bell' by flashing the entire screen. The Sun-2 models have an audible bell which beeps. The window system flashes the window.

control–H (0x8) Backspace (BS)

The cursor moves one position to the left on the current line. If it is already at the left edge of the screen, nothing happens.

control–I (0x9) Tab (TAB)

The cursor moves right on the current line to the next tab stop. The tab stops are fixed at every multiple of 8 columns. If the cursor is already at the right edge of the screen, nothing happens; otherwise the cursor moves right a minimum of one and a maximum of eight character positions.

control–J (0xA) Line-feed (LF)

The cursor moves down one line, remaining at the same character position on the line. If the cursor is already at the bottom line, the screen either scrolls up or "wraps around" depending on the setting of an internal variable *S* (initially 1) which can be changed by the ESC[r control sequence. If *S* is greater than zero, the entire screen (including the cursor) is scrolled up by *S* lines before executing the line-feed. The top *S* lines scroll off the screen and are lost. *S* new blank lines scroll onto the bottom of the screen. After scrolling, the line-feed is executed by moving the cursor down one line.

If *S* is zero, 'wrap-around' mode is entered. 'ESC [ 1 r' exits back to scroll mode. If a line-feed occurs on the bottom line in wrap mode, the cursor goes to the same character position in the top line of the screen. When any line-feed occurs, the line that the cursor moves to is cleared. This means that no scrolling occurs. Wrap-around mode is not implemented in the window system.

The screen scrolls as fast as possible depending on how much data is backed up waiting to be printed. Whenever a scroll must take place and the console is in normal scroll mode ('ESC [ 1 r'), it scans the rest of the data awaiting printing to see how many line-feeds occur in it. This scan stops when any control character from the set {**VT, FF, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US**} is found. At that point, the screen is scrolled by N lines ($N \geq 1$) and processing continues. The scanned text is still processed normally to fill in the newly created lines. This results in much faster scrolling with scrolling as long as no escape codes or other control characters are intermixed with the text.

See also the discussion of the 'Set scrolling' (**ESC[r**) control function below.

control–K (0xB) Reverse Line-feed

The cursor moves up one line, remaining at the same character position on the line. If the cursor is already at the top line, nothing happens.

control–L (0xC) Form-feed (FF)

The cursor is positioned to the Home position (upper-left corner) and the entire screen is cleared.

control–M (0xD) Return (CR)

The cursor moves to the leftmost character position on the current line.

**Escape Sequence Functions**

control–[ (0x1B) Escape (ESC)

This is the escape character. Escape initiates a multi-character control sequence.

ESC[#@ Insert Character (ICH)

Takes one parameter, # (default 1). Inserts # spaces at the current cursor position. The tail of the current line starting at the current cursor position inclusive is shifted to the right by # character positions to make room for the spaces. The rightmost # character positions shift off the line and are lost. The position of the cursor is unchanged.

ESC[#A Cursor Up (CUU)

Takes one parameter, # (default 1). Moves the cursor up # lines. If the cursor is fewer than # lines from the top of the screen, moves the cursor to the topmost line on the screen. The character position of the cursor on the line is unchanged.

ESC[#B Cursor Down (CUD)

Takes one parameter, # (default 1). Moves the cursor down # lines. If the cursor is fewer than # lines from the bottom of the screen, move the cursor to the last line on the screen. The character position of the cursor on the line is unchanged.

ESC[#C Cursor Forward (CUF)

Takes one parameter, # (default 1). Moves the cursor to the right by # character positions on the current line. If the cursor is fewer than # positions from the right edge of the screen, moves the cursor to the rightmost position on the current line.

ESC[#D Cursor Backward (CUB)

Takes one parameter, # (default 1). Moves the cursor to the left by # character positions on the current line. If the cursor is fewer than # positions from the left edge of the screen, moves the cursor to the leftmost position on the current line.

ESC[#E Cursor Next Line (CNL)

Takes one parameter, # (default 1). Positions the cursor at the leftmost character position on the #-th line below the current line. If the current line is less than # lines from the bottom of the screen, positions the cursor at the leftmost character position on the bottom line.

ESC[#1;#2f Horizontal And Vertical Position (HVP)

or

ESC[#1;#2H Cursor Position (CUP)

Takes two parameters, #1 and #2 (default 1, 1). Moves the cursor to the #2-th character position on the #1-th line. Character positions are numbered from 1 at the left edge of the screen; line positions are numbered from 1 at the top of the screen. Hence, if both parameters are omitted, the default action moves the cursor to the home position (upper left corner). If only one parameter is supplied, the cursor moves to column 1 of the specified line.

ESC[J Erase in Display (ED)

Takes no parameters. Erases from the current cursor position inclusive to the end of the screen. In other words, erases from the current cursor position inclusive to the end of the current line and all lines below the current line. The cursor position is unchanged.

ESC[K Erase in Line (EL)

Takes no parameters. Erases from the current cursor position inclusive to the end of the current line. The cursor position is unchanged.

ESC[#L Insert Line (IL)

Takes one parameter, # (default 1). Makes room for # new lines starting at the current line by scrolling down by # lines the portion of the screen from the current line inclusive to the bottom. The # new lines at the cursor are filled with spaces; the bottom # lines shift off the bottom of the screen and are lost. The position of the cursor on the screen is unchanged.

ESC[#M Delete Line (DL)

Takes one parameter, # (default 1). Deletes # lines beginning with the current line. The portion of the screen from the current line inclusive to the bottom is scrolled upward by # lines. The # new lines scrolling onto the bottom of the screen are filled with spaces; the # old lines beginning at the cursor line are deleted. The position of the cursor on the screen is unchanged.

ESC[#P Delete Character (DCH)

Takes one parameter, # (default 1). Deletes # characters starting with the current cursor position. Shifts to the left by # character positions the tail of the current line from the current cursor position inclusive to the end of the line. Blanks are shifted into the rightmost # character positions. The position of the cursor on the screen is unchanged.

ESC[#m Select Graphic Rendition (SGR)

Takes one parameter, # (default 0). Note: unlike most escape sequences, the parameter defaults to zero if omitted. Invokes the graphic rendition specified by the parameter. All following printing characters in the data stream are rendered according to the parameter until the next occurrence of this escape sequence in the data stream. Currently only two graphic renditions are defined:

0 Normal rendition.

7 Negative (reverse) image.

Negative image displays characters as white-on-black if the screen mode is currently black-on white, and vice-versa. Any non-zero value of # is currently equivalent to 7 and selects the negative image rendition.

ESC[p Black On White (SUNBOW)

Takes no parameters. Sets the screen mode to black-on-white. If the screen mode is already black-on-white, has no effect. In this mode spaces display as solid white, other characters as black-on-white. The cursor is a solid black block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is white-on-black in this mode. This is the initial setting of the screen mode on reset.

ESC[q White On Black (SUNWOB)

Takes no parameters. Sets the screen mode to white-on-black. If the screen mode is already white-on-black, has no effect. In this mode spaces display as solid black, other characters as white-on-black. The cursor is a solid white block. Characters displayed in negative image rendition (see 'Select Graphic Rendition' above) is black-on-white in this mode. The initial setting of the screen mode on reset is the alternative mode, black on white.

ESC[#r Set scrolling (SUNSCRL)

Takes one parameter, # (default 0). Sets to # an internal register which determines how many lines the screen scrolls up when a line-feed function is performed with the cursor on the bottom line. A parameter of 2 or 3 introduces a small amount of "jump" when a scroll occurs. A parameter of 34 clears the screen rather than scrolling. The initial setting is 1 on reset.

A parameter of zero initiates "wrap mode" instead of scrolling. In wrap mode, if a linefeed occurs on the bottom line, the cursor goes to the same character position in the top line of the screen. When any linefeed occurs, the line that the cursor moves to is cleared. This means that no scrolling ever occurs. 'ESC [ 1 r' exits back to scroll mode.

For more information, see the description of the Line-feed (CTRL-J) control function above.

ESC[s Reset terminal emulator (SUNRESET)

Takes no parameters. Resets all modes to default, restores current font from PROM. Screen and cursor position are

**4014 TERMINAL EMULATION**

The PROM monitor for Sun models 100U and 150U provides the Sun Workstation with the capability to emulate a subset of the Tektronix 4014 terminal. This feature does not exist in other Sun PROMs and will be removed from models 100U and 150U in future Sun releases. **tektool**(1) provides Tektronix 4014 terminal emulation and should be used instead of relying on the capabilities of the PROM monitor.

**FILES**

**/dev/console**

**SEE ALSO**

**tektool**(1) **kb**(4M), **pty**(4), **termio**(4), **ttcompat**(4M), **ldterm**(4M), **win**(4S), **zs**(4S)

ANSI Standard X3.64, "*Additional Controls for Use with ASCII*", Secretariat: CBEMA, 1828 L St., N.W., Washington, D.C. 20036.

**BUGS**

**TIOCCONS** should be restricted to the owner of **/dev/console**.

**NAME**

des – DES encryption chip interface

**CONFIG — SUN-3 SYSTEM**

**des0 at obio ? csr 0x1c0000**

**#include <sys/des.h>**

**CONFIG — SUN-2 SYSTEM**

**des0 at virtual ? csr 0xee1800**

**#include <sys/des.h>**

**DESCRIPTION**

The **des** driver provides a high level interface to the AmZ8068 Data Ciphering Processor, a hardware implementation of the NBS Data Encryption Standard.

The high level interface provided by this driver is hardware independent and could be shared by future drivers in other systems.

The interface allows access to two modes of the DES algorithm: Electronic Code Book (ECB) and Cipher Block Chaining (CBC). All access to the DES driver is through **ioctl**(2) calls rather than through reads and writes; all encryption is done in-place in the user's buffers.

**IOCTLS**

The ioctls provided are:

**DESIOCBLOCK**

This call encrypts/decrypts an entire buffer of data, whose address and length are passed in the '**struct desparams**' addressed by the argument. The length must be a multiple of 8 bytes.

**DESIOCQUICK**

This call encrypts/decrypts a small amount of data quickly. The data is limited to **DES_QUICKLEN** bytes, and must be a multiple of 8 bytes. Rather than being addresses, the data is passed directly in the '**struct desparams**' argument.

**FILES**

**/dev/des**

**SEE ALSO**

**des**(1), **des_crypt**(3)

*Federal Information Processing Standards Publication 46*

*AmZ8068 DCP Product Description, Advanced Micro Devices*

**NAME**

dkio – generic disk control operations

**DESCRIPTION**

All Sun disk drivers support a set of ioctl's for disk formatting and labeling operations. Basic to these ioctl's are the definitions in **<sun/dkio.h>**:

```
/*
* Structures and definitions for disk io control commands
*/
/* Disk identification */
struct dk_info {
        int     dki_ctlr;               /* controller address */
        short   dki_unit;               /* unit (slave) address */
        short   dki_ctype;              /* controller type */
        short   dki_flags;              /* flags */
};
/* controller types */
#define DKC_UNKNOWN             0
#define DKC_SMD2180             1
#define DKC_DSD5215             5
#define DKC_XY450               6
#define DKC_ACB4000             7
#define DKC_MD21                8
#define DKC_CSS         12
#define DKC_NEC765      13      /* floppy on Sun386i */
/* flags */
#define DKI_BAD144      0x01    /* use DEC std 144 bad sector fwding */
#define DKI_MAPTRK      0x02    /* controller does track mapping */
#define DKI_FMTTRK      0x04    /* formats only full track at a time */
#define DKI_FMTVOL      0x08    /* formats only full volume at a time */
/* Definition of a disk's geometry */
struct dk_geom {
        unsigned short  dkg_ncyl;       /* # of data cylinders */
        unsigned short  dkg_acyl;       /* # of alternate cylinders */
        unsigned short  dkg_bcyl;       /* cyl offset (for fixed head area) */
        unsigned short  dkg_nhead;      /* # of heads */
        unsigned short  dkg_bhead;      /* head offset (for Larks, etc.) */
        unsigned short  dkg_nsect;      /* # of sectors per track */
        unsigned short  dkg_intrlv;     /* interleave factor */
        unsigned short  dkg_gap1;       /* gap 1 size */
        unsigned short  dkg_gap2;       /* gap 2 size */
        unsigned short  dkg_apc;        /* alternates per cyl (SCSI only) */
        unsigned short  dkg_extra[9];   /* for compatible expansion */
};
/* disk io control commands */
#define DKIOCGGEOM  _IOR(d, 2, struct dk_geom)      /* Get geometry */
#define DKIOCSGEOM          _IOW(d, 3, struct dk_geom)      /* Set geometry */
#define DKIOCGPART          _IOR(d, 4, struct dk_map)       /* Get partition info */
#define DKIOCSPART          _IOW(d, 5, struct dk_map)       /* Set partition info */
#define DKIOCINFO           _IOR(d, 8, struct dk_info)      /* Get info */
```

The **DKIOCINFO** ioctl returns a **dk_info** structure which tells the type of the controller and attributes about how bad-block processing is done on the controller. The **DKIOCGPART** and **DKIOCSPART** get and set the controller's current notion of the partition table for the disk (without changing the partition table on the

disk itself), while the **DKIOCGGEOM** and **DKIOCSGEOM** ioctl's do similar things for the per-drive geometry information.

**SEE ALSO**

**ip**(4P), **sd**(4S), **xy**(4S)

**NAME**

drum – paging device

**CONFIG**

None; included with standard system.

**SYNOPSIS**

**#include <fcntl.h>**

**open("/dev/drum",** *mode*);

**DESCRIPTION**

This file refers to the paging device in use by the system. This may actually be a subdevice of one of the disk drivers, but in a system with paging interleaved across multiple disk drives it provides an indirect driver for the multiple drives.

**FILES**

**/dev/drum**

**BUGS**

Reads from the drum are not allowed across the interleaving boundaries. Since these only occur every .5Mbytes or so, and since the system never allocates blocks across the boundary, this is usually not a problem.

## NAME

ec – 3Com 10 Mb/s Ethernet interface

## CONFIG — SUN-2 SYSTEM

**device ec0 at mbmem ? csr 0xe0000 priority 3**
**device ec1 at mbmem ? csr 0xe2000 priority 3**

## DESCRIPTION

The **ec** interface provides access to a 10 Mb/s Ethernet network through a 3COM controller. For a general description of network interfaces see **if**(4N).

The hardware consumes 8 kilobytes of Multibus memory space. This memory is used for internal buffering by the board. The board starts at standard addresses 0xE0000 or 0xE2000. The board must be configured for interrupt level 3.

The interface software implements an exponential backoff algorithm when notified of a collision on the cable.

The interface handles the Internet protocol family, with the interface address maintained in Internet format. The Address Resolution Protocol **arp**(4P) is used to map 32-bit Internet addresses used in **inet**(4F) to the 48-bit addresses used on the Ethernet.

## DIAGNOSTICS

**ec%d: Ethernet jammed**
After 16 failed transmissions and backoffs using the exponential backoff algorithm, the packet was dropped.

**ec%d: can't handle af%d**
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

## SEE ALSO

**arp**(4P), **if**(4N), **inet**(4F)

## BUGS

The interface hardware is not capable of talking to itself, making diagnosis more difficult.

**NAME**

fb – driver for Sun console frame buffer

**CONFIG**

None; included in standard system.

**DESCRIPTION**

The **fb** driver provides indirect access to a Sun graphics controller board. It is an indirect driver for the Sun workstation console's frame buffer. At boot time, the workstation's frame buffer device is determined from information from the PROM monitor and set to be the one that **fb** will indirect to. The device driver for the console's frame buffer must be configured into the kernel so that this indirect driver can access it.

The idea behind this driver is that user programs can open a known device, query its characteristics and access it in a device dependent way, depending on the type. **fb** redirects **open**(2V), **close**(2), **ioctl**(2), and **mmap**(2) calls to the real frame buffer. All of the Sun frame buffers support the same general interface; see **fbio**(4S)

**FILES**

**/dev/fb**

**SEE ALSO**

**close**(2), **ioctl**(2), **mmap**(2), **open**(2V), **bwone**(4S), **bwtwo**(4S), **cgone**(4S), **cgtwo**(4S), **fbio**(4S), **gpone**(4S)

**NAME**

fbio – general properties of frame buffers

**DESCRIPTION**

All of the Sun frame buffers support the same general interface. Each responds to a **FBIOGTYPE** ioctl which returns information in a structure defined in **<sun/fbio.h>**:

```
struct  fbtype {
        int     fb_type;        /* as defined below */
        int     fb_height;      /* in pixels */
        int     fb_width;       /* in pixels */
        int     fb_depth;       /* bits per pixel */
        int     fb_cmsize;      /* size of color map (entries) */
        int     fb_size;        /* total size in bytes */
};

#define FBTYPE_SUN1BW           0
#define FBTYPE_SUN1COLOR        1
#define FBTYPE_SUN2BW           2
#define FBTYPE_SUN2COLOR        3
#define FBTYPE_SUN2GP           4
#define FBTYPE_SUN3COLOR        6
#define FBTYPE_SUN4COLOR        8
```

Each device has an **FBTYPE** which is used by higher-level software to determine how to perform raster-op and other functions. Each device is used by opening it, doing an **FBIOGTYPE** ioctl to see which frame buffer type is present, and thereby selecting the appropriate device-management routines.

Full-fledged frame buffers (that is, those that run SunView) implement an **FBIOGPIXRECT** ioctl, which returns a pixrect. This call is made only from inside the kernel. The returned pixrect is used by **win**(4S) for cursor tracking and colormap loading.

**FBIOSVIDEO** and **FBIOGVIDEO** are general-purpose ioctls for controlling possible video features of frame buffers. They are defined in **<sun/fbio.h>**. These ioctls either set or return the value of a flags integer. At this point, only the **FBVIDEO_ON** option is available, controlled by **FBIOSVIDEO**. **FBIOGVIDEO** returns the current video state.

The **FBIOSATTR** and **FBIOGATTR** ioctls allow access to special features of newer frame buffers. They use the following structures as defined in **<sun/fbio.h>**:

```
#define FB_ATTR_NDEVSPECIFIC    8       /* no. of device specific values */
#define FB_ATTR_NEMUTYPES 4     /* no. of emulation types */
struct fbsattr {
        int     flags;                  /* misc flags */
#define FB_ATTR_AUTOINIT        1       /* emulation auto init flag */
#define FB_ATTR_DEVSPECIFIC 2   /* dev. specific stuff valid flag */
        int     emu_type;               /* emulation type (-1 if unused) */
        int     dev_specific[FB_ATTR_NDEVSPECIFIC];  /* catchall */
};
struct fbgattr {
        int     real_type;              /* real device type */
        int     owner;                  /* PID of owner, 0 if myself */
        struct fbtype fbtype;           /* fbtype info for real device */
        struct fbsattr sattr;           /* see above */
        int     emu_types[FB_ATTR_NEMUTYPES];   /* possible emulations */
                                        /* (-1 if unused) */
};
```

**SEE ALSO**

**mmap**(2), **bwone**(4S), **bwtwo**(4S), **cgfour**(4S), **cgone**(4S), **cgtwo**(4S), **fb**(4S), **gpone**(4S), **win**(4S)

**BUGS**

**FBIOSATTR** and **FBIOGATTR** are only supported by the **cgfour**(4S) frame buffer.

The **FBVIDEO_ON** flag my be incorrect for Sun-1 system black and white frame buffers; see **bwone**(4S).

## NAME

fd – Disk driver for Floppy Disk Controllers

## CONFIG

**controller fdc0 at atmem ? csr 0x1000 dmachan 2 irq 6 priority 2**
**disk fd0 at fdc0 drive 0 flags 0**

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

The block-files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

### Disk Support

The **fd0** partition on a floppy disk is normally used for the file system.

## FILES

1.44 MB Floppy Disk Drives:

| | |
|---|---|
| **/dev/fd0a** | block file |
| **/dev/fd0c** | block file **/dev/rfd0a** raw file **/dev/rfd0c** raw file |

720 K Floppy Disk Drives:

| | |
|---|---|
| **/dev/fdl0a** | block file |
| **/dev/fdl0c** | block file |
| **/dev/rfdl0a** | raw file |
| **/dev/rfdl0c** | raw file |

## SEE ALSO

**dkio**(4S)

## DIAGNOSTICS

**fd drv %d, trk %d: %s**
A command such as read or write encountered a format-related error condition. The value of *%s* is derived from the error number given by the controller, indicating the nature of the error. The track number is relative to the beginning of the partition involved.

**fd drv %d, blk %d: %s**
A command such as read or write encountered an error condition related to I/O. The value of *%s* is derived from the error number returned by the controller and indicates the nature of the error. The block number is relative to the start of the partition involved.

**fd controller: %s**
An error occurred in the controller. The value of *%s* is derived from the status returned by the controller and specifies the error encountered.

**fd(%d): %s please insert**
I/O was attempted while the floppy drive door was not latched. The value of *%s* indicates which disk was expected to be in the drive.

## NAME

filio – ioctls that operate directly on files, file descriptors, and sockets

## SYNOPSIS

**#include <sys/filio.h>**

## DESCRIPTION

The IOCTL's listed in this manual page apply directly to files, file descriptors, and sockets, independent of any underlying device or protocol.

Note: the **fcntl**(2V) system call is the primary method for operating on file descriptors as such, rather than on the underlying files.

### IOCTLS for File Descriptors

**FIOCLEX** The argument is ignored. Set the close-on-exec flag for the file descriptor passed to **ioctl**. This flag is also manipulated by the **F_SETFD** command of **fcntl**(2V).

**FIONCLEX** The argument is ignored. Clear the close-on-exec flag for the file descriptor passed to **ioctl**.

### IOCTLs for Files

**FIONREAD** The argument is a pointer to a **long**. Set the value of that **long** to the number of immediately readable characters from whatever the descriptor passed to **ioctl** refers to. This works for files, pipes, sockets, and terminals.

**FIONBIO** The argument is a pointer to an **int**. Set or clear non-blocking I/O. If the value of that **int** is a 1 (one) the descriptor is set for non-blocking I/O. If the value of that **int** is a 0 (zero) the descriptor is cleared for non-blocking I/O.

**FIOASYNC** The argument is a pointer to an **int**. Set or clear asynchronous I/O. If the value of that **int** is a 1 (one) the descriptor is set for asynchronous I/O. If the value of that **int** is a 0 (zero) the descriptor is cleared for asynchronous I/O.

**FIOSETOWN** The argument is a pointer to an **int**. Set the process-group ID that will subsequently receive **SIGIO** or **SIGURG** signals for the object referred to by the descriptor passed to **ioctl** to the value of that **int**.

**FIOGETOWN** The argument is a pointer to an **int**. Set the value of that **int** to the process-group ID that is receiving **SIGIO** or **SIGURG** signals for the object referred to by the descriptor passed to **ioctl**.

## SEE ALSO

**ioctl**(2), **fcntl**(2V), **getsockopt**(2), **sockio**(4)

**NAME**

fpa – Sun-3 floating-point accelerator

**CONFIG — SUN-3 SYSTEM**

**fpa0 at virtual ? csr 0xe0000000**

**DESCRIPTION**

The **fpa** is a floating point accelerator available for Sun-3 systems.

The **fpa** device driver manipulates the 32 contexts supported by the floating point accelerator hardware.

The **open**(2V), **close**(2), and **ioctl**(2) system calls generally produce errors when applied to this device. However, since the 32 **fpa** contexts are allocated and deallocated automatically, and no user program needs to access **fpa** registers explicitly, such calls to the device are generally unnecessary. Access to the device is normally provided at compile time by a compiler option, such as the **–ffpa** option to **cc**(1V).

**FILES**

**/dev/fpa**

**SEE ALSO**

**cc**(1V), **close**(2), **ioctl**(2), **open**(2V)

**NAME**

gpone – Sun-3/Sun-2 graphics processor

**CONFIG — SUN-3 SYSTEM**

**device gpone0 at vme24d16 ? csr**

**CONFIG — SUN-2 SYSTEM**

**device gpone0 at vme24 ? csr**

**DESCRIPTION**

The **gpone** interface provides access to the optional Graphics Processor Board (GP).

The hardware consumes 64 kilobytes of VME bus address space. The GP board starts at standard address 0x210000 and must be configured for interrupt level 3.

**IOCTLS**

The graphics processor responds to a number of ioctl calls as described here. One of the calls uses a **gp1fbinfo** structure that looks like this:

```
struct  gp1fbinfo {
        int             fb_vmeaddr;     /* physical color board address */
        int             fb_hwwidth;     /* fb board width */
        int             fb_hwheight;    /* fb board height */
        int             addrdelta;      /* phys addr diff between fb and gp */
        caddr_t         fb_ropaddr;     /* cg2 va thru kernelmap */
        int             fbunit;         /* fb unit to use  for a,b,c,d */
};
```

The ioctl call looks like this:

```
ioctl(file, request, argp)
int file, request;
```

**argp** is defined differently for each GP ioctl request and is specified in the descriptions below.

The following ioctl commands provide for transferring data between the graphics processor and color boards and processes.

**GP1IO_PUT_INFO**

Passes information about the frame buffer into driver. **argp** points to a **struct gp1fbinfo** which is passed to the driver.

**GP1IO_GET_STATIC_BLOCK**

Hands out a static block from the GP. **argp** points to an **int** which is returned from the driver.

**GP1IO_FREE_STATIC_BLOCK**

Frees a static block from the GP. **argp** points to an **int** which is passed to the driver.

**GP1IO_GET_GBUFFER_STATE**

Checks to see if there is a buffer present on the GP. **argp** points to an **int** which is returned from the driver.

**GP1IO_CHK_GP**

Restarts the GP if necessary. **argp** points to an **int** which is passed to the driver.

**GP1IO_GET_RESTART_COUNT**

Returns the number of restarts of a GP since power on. Needed to differentiate **SIGXCPU** calls in user processes. **argp** points to an **int** which is returned from the driver.

**GP1IO_REDIRECT_DEVFB**

Configures **/dev/fb** to talk to a graphics processor device. **argp** points to an **int** which is passed to the driver.

**GP1IO_GET_REQDEV**

Returns the requested minor device. **argp** points to a **dev_t** which is returned from the driver.

**GP1IO_GET_TRUMINORDEV**
Returns the true minor device. **argp** points to a **char** which is returned from the driver.

The graphics processor driver also responds to the **FBIOGTYPE**, ioctl which a program can use to inquire as to the characteristics of the display device, the **FBIOGINFO**, ioctl for passing generic information, and the **FBIOGPIXRECT** ioctl so that SunWindows can run on it. See **fbio**(4S).

## FILES

**/dev/gpone[0-3][abcd]**
**/usr/include/sun/gpio.h**
**/usr/include/pixrect/{gp1cmds.h,gp1reg.h,gp1var.h}**
**/dev/fb**

## SEE ALSO

**fbio**(4S), **mmap**(2), **gpconfig**(8)

*SunCGI Reference Manual*

## DIAGNOSTICS

**The Graphics Processor has been restarted. You may see display garbage as a result.**

## NAME

icmp – Internet Control Message Protocol

## SYNOPSIS

**#include <sys/socket.h>**
**#include <netinet/in.h>**
**#include <netinet/ip_icmp.h>**

**s = socket(AF_INET, SOCK_RAW, proto);**

## DESCRIPTION

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed through a "raw socket" for network monitoring and diagnostic functions. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from **getprotobyname** (see **getprotoent**(3N)). ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls, though the **connect**(2) call may also be used to fix the destination for future packets (in which case the **read**(2V) or **recv**(2) and **write**(2V) or **send**(2) system calls may be used).

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the holder of a raw socket with the IP header and options intact.

ICMP is an unreliable datagram protocol layered above IP. It is used internally by the protcol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP "redirect" message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided using the ICMP raw socket.

## ERRORS

A socket operation may fail with one of the following errors returned:

| | |
|---|---|
| EISCONN | when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected; |
| ENOTCONN | when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected; |
| ENOBUFS | when the system runs out of memory for an internal data structure; |
| EADDRNOTAVAIL | when an attempt is made to create a socket with a network address for which no network interface exists. |

## SEE ALSO

**connect**(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **getprotoent**(3N), **inet**(4F), **ip**(4P), **routing**(4N)

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1064-01)

## BUGS

Replies to ICMP "echo" messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

## NAME

ie – Intel 10 Mb/s Ethernet interface

## CONFIG — SUN-3 SYSTEM

**device ie0 at obio ? csr 0xc0000 priority 3**
**device ie1 at vme24d16 ? csr 0xe88000 priority 3 vector ieintr 0x75**
**device ie1 at vme24d16 ? csr 0x31ff02 priority 3 vector ieintr 0x75**

## CONFIG — SUN-2 SYSTEM

**device ie0 at obio 2 csr 0x7f0800 priority 3**
**device ie1 at vme24 ? csr 0xe88000 priority 3 vector ieintr 0x75**
**device ie0 at mbmem ? csr 0x88000 priority 3**
**device ie1 at mbmem ? csr 0x8c000 flags 2 priority 3**

## CONFIG — Sun386i SYSTEM

**device ie0 at obmem ? csr 0xD0000000 irq 21 priority 3**

## DESCRIPTION

The **ie** interface provides access to a 10 Mb/s Ethernet network through the Intel 82586 controller chip. For a general description of network interfaces see **if**(4N).

In the Sun-3 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface. The second line specifies a Multibus Intel Ethernet interface for use with a VME adapter. The third line specifies the Intel Ethernet interface present on a Sun-3 Eurocard board.

In the Sun-2 lines above, the first line specifies the CPU-board-resident Intel Ethernet interface on a Sun-2/50 or Sun-2/160 system. The second line specifies a Multibus Intel Ethernet controller for use with a VME adapter on these systems. The third line specifies the first Multibus Intel Ethernet controller for a Sun-2/120 or Sun-2/170 system. The fourth line specifies the second such controller for these systems.

The Sun386i line above specifies the CPU-board-resident Intel Ethernet interface.

## SEE ALSO

**if**(4N)

## DIAGNOSTICS

There are too many driver messages to list them all individually here. Some of the more common messages and their meanings follow.

**ie%d: Ethernet jammed**
Network activity has become so intense that sixteen successive transmission attempts failed, and the 82586 gave up on the current packet. Another possible cause of this message is a noise source somewhere in the network, such as a loose transceiver connection.

**ie%d: no carrier**
The 82586 has lost input to its carrier detect pin while trying to transmit a packet, causing the packet to be dropped. Possible causes include an open circuit somewhere in the network and noise on the carrier detect line from the transceiver.

**ie%d: lost interrupt: resetting**
The driver and 82586 chip have lost synchronization with each other. The driver recovers by resetting itself and the chip.

**ie%d: iebark reset**
The 82586 failed to complete a watchdog timeout command in the alloted time. The driver recovers by resetting itself and the chip.

**ie%d: WARNING: requeueing**
The driver has run out of resources while getting a packet ready to transmit. The packet is put back on the output queue for retransmission after more resources become available.

**ie%d: panic: scb overwritten**

The driver has discovered that memory that should remain unchanged after initialization has become corrupted. This error usually is a symptom of a bad 82586 chip.

**NAME**

if – general properties of network interfaces

**DESCRIPTION**

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, **lo**(4), do not.

At boot time, each interface with underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address, it is expected to install a routing table entry so that messages can be routed through it. Most interfaces require some part of their address specified with an **SIOCSIFADDR** IOCTL before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (for example, 10Mb/s Ethernets using **arp**(4P),), the entire address specified in the ioctl is used.

The following **ioctl** calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an **ifreq** structure as its parameter. This structure has the form

```
struct  ifreq {
        char    ifr_name[16];           /* name of interface (e.g. "ec0") */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                short   ifru_flags;
        } ifr_ifru;
#define ifr_addr                ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_flags       ifr_ifru.ifru_flags     /* flags */
};
```

**SIOCSIFADDR** Set interface address. Following the address assignment, the "initialization" routine for the interface is called.

**SIOCGIFADDR** Get interface address.

**SIOCSIFDSTADDR** Set point to point address for interface.

**SIOCGIFDSTADDR** Get point to point address for interface.

**SIOCSIFFLAGS** Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

**SIOCGIFFLAGS** Get interface flags.

**SIOCGIFCONF** Get interface configuration list. This request takes an **ifconf** structure (see below) as a value-result parameter. The **ifc_len** field should be initially set to the size of the buffer pointed to by **ifc_buf**. On return it will contain the length, in bytes, of the configuration list.

```
/*
* Structure used in SIOCGIFCONF request.
* Used to retrieve interface configuration
* for machine (useful for programs which
* must know all networks accessible).
*/
struct  ifconf {
        int     ifc_len;        /* size of associated buffer */
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
#define ifc_buf  ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req  ifc_ifcu.ifcu_req /* array of structures returned */
};
```

**SIOCADDMULTI** Enable a multicast address for the interface. A maximum of 64 multicast addresses may be enabled for any given interface.

**SIOCDELMULTI** Disable a previously set multicast address.

**SIOCSPROMISC** Toggle promiscuous mode.

**SEE ALSO**

**arp**(4P), **ec**(4S), **lo**(4)

## NAME

inet – Internet protocol family

## SYNOPSIS

**options INET**

```
#include <sys/types.h>
#include <netinet/in.h>
```

## DESCRIPTION

The Internet protocol family implements a collection of protocols which are centered around the *Internet Protocol* (IP) and which share a common address format. The Internet family provides protocol support for the **SOCK_STREAM**, **SOCK_DGRAM**, and **SOCK_RAW** socket types.

## PROTOCOLS

The Internet protocol family is comprised of the Internet Protocol (IP), the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP is used to support the **SOCK_STREAM** abstraction while UDP is used to support the **SOCK_DGRAM** abstraction; see **tcp**(4P) and **udp**(4P). A raw interface to IP is available by creating an Internet socket of type **SOCK_RAW**; see **ip**(4P). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs; see **icmp**(4P). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses; see **arp**(4P).

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded; the most-significant bit is zero in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Sites with a cluster of local networks may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet number and host number parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following **ioctl**(2) commands on a datagram socket in the Internet domain; they have the same form as the **SIOCIFADDR** command (see **intro**(4N)).

**SIOCSIFNETMASK** Set interface network mask. The network mask defines the network part of the address; if it contains more of the address than the address type would indicate, then subnets are in use.

**SIOCGIFNETMASK** Get interface network mask.

## ADDRESSING

IP addresses are four byte quantities, stored in network byte order (on Sun386i systems these are word and byte reversed).

Sockets in the Internet protocol family use the following addressing structure:

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

Library routines are provided to manipulate structures of this form; see **intro**(3N).

The **sin_addr** field of the **sockaddr_in** structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value **INADDR_ANY** may be used in this field to effect "wildcard" matching. Given in a **bind**(2) call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP

address is or when a process wishes to receive requests using all of its network interfaces. The **sockaddr_in** structure given in the **bind**(2) call must specify an **in_addr** value of either **IPADDR_ANY** or one of the system's valid IP addresses. Requests to bind any other address will elicit the error EADDRNOTAVAIL. When a **connect**(2) call is made for a socket that has a wildcard local address, the system sets the **sin_addr** field of the socket to the IP address of the network interface that the packets for that connection are routed via.

The **sin_port** field of the **sockaddr_in** structure specifies a port number used by TCP or UDP. The local port address specified in a **bind**(2) call is restricted to be greater than **IPPORT_RESERVED** (defined in **<netinet/in.h>**) unless the creating process is running as the super-user, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error EADDRINUSE. If the local port address is specified as 0, then the system picks a unique port address greater than **IPPORT_RESERVED**. A unique local port address is also picked when a socket which is not bound is used in a **connect**(2) or **sendto** (see **send**(2)) call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling **socket**(2) and then **connect**(2), and to send UDP datagrams with a **socket**(2) call followed by a **sendto**(2) call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the **SO_REUSEADDR** socket option with **setsockopt** (see **getsockopt**(2)).

## SEE ALSO

**bind**(2), **connect**(2), **getsockopt**(2), **ioctl**(2), **sendto**(2), **socket**(2), **byteorder**(3N), **gethostent**(3N), **getnetent**(3N), **getprotoent**(3N), **getservent**(3N), **inet**(3N), **intro**(3N), **arp**(4P), **icmp**(4P), **intro**(4N), **ip**(4P) **tcp**(4P), **udp**(4P),

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information Center, SRI International, Menlo Park, Calif., 1985.
*A 4.2BSD Interprocess Communication Primer*

## CAVEAT

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

**NAME**

ip – Internet Protocol

**SYNOPSIS**

**#include <sys/socket.h>**
**#include <netinet/in.h>**

**s = socket(AF_INET, SOCK_RAW, proto);**

**DESCRIPTION**

IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly using a "raw socket." See **tcp**(4P) and **udp**(4P). The protocol options defined in the IP specification may be set in outgoing datagrams.

Raw IP sockets are connectionless and are normally used with the **sendto** and **recvfrom** calls, (see **send**(2) and **recv**(2)) although the **connect**(2) call may also be used to fix the destination for future datagrams (in which case the **read**(2V) or **recv**(2) and **write**(2V) or **send**(2) calls may be used). If **proto** is zero, the default protocol, **IPPROTO_RAW**, is used. If **proto** is non-zero, that protocol number will be set in outgoing datagrams and will be used to filter incoming datagrams. An IP header will be generated and prepended to each outgoing datagram; Received datagrams are returned with the IP header and options intact.

A single socket option, **IP_OPTIONS**, is supported at the IP level. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with **setsockopt** (see **getsockopt**(2)). The **getsockopt**(2) call returns the IP options set in the last **setsockopt** call. IP options on received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in **setsockopt** matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

At the socket level, the socket option **SO_DONTROUTE** may be applied. This option forces datagrams being sent to bypass the routing step in output. Normally, IP selects a network interface to send the datagram via, and possibly an intermediate gateway, based on an entry in the routing table. See **routing**(4N). When **SO_DONTROUTE** is set, the datagram will be sent via the interface whose network number or full IP address matches the destination address. If no interface matches, the error ENETUNRCH will be returned.

Datagrams flow through the IP layer in two directions: from the network **ip** to user processes and from user processes *down* to the network. Using this orientation, IP is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMP) is logically a part of IP. See **icmp**(4P).

IP provides for a checksum of the header part, but not the data part of the datagram. The checksum value is computed and set in the process of sending datagrams and checked when receiving datagrams. IP header checksumming may be disabled for debugging purposes by patching the kernel variable **ipcksum** to have the value zero.

IP options in received datagrams are processed in the IP layer according to the protocol specification. Currently recognized IP options include: security, loose source and record route (LSRR), strict source and record route (SSRR), record route, stream identifier, and internet timestamp.

The IP layer will normally forward received datagrams that are not addressed to it. Forwarding is under the control of the kernel variable *ipforwarding*: if *ipforwarding* is zero, IP datagrams will not be forwarded; if *ipforwarding* is one, IP datagrams will be forwarded. *ipforwarding* is usually set to one only in machines with more than one network interface (internetwork routers). This kernel variable can be patched to enable or disable forwarding.

The IP layer will send an ICMP message back to the source host in many cases when it receives a datagram that can not be handled. A "time exceeded" ICMP message will be sent if the "time to live" field in the IP header drops to zero in the process of forwarding a datagram. A "destination unreachable" message will be sent if a datagram can not be forwarded because there is no route to the final destination, or if it can not be fragmented. If the datagram is addressed to the local host but is destined for a protocol that is not supported or a port that is not in use, a destination unreachable message will also be sent. The IP layer may send an ICMP "source quench" message if it is receiving datagrams too quickly. ICMP messages are only sent for the first fragment of a fragmented datagram and are never returned in response to errors in other ICMP messages.

The IP layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IP back up to the user process.

## ERRORS

A socket operation may fail with one of the following errors returned:

| | |
|---|---|
| EACCESS | when specifying an IP broadcast destination address if the caller is not the superuser; |
| EISCONN | when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected; |
| EMSGSIZE | when sending datagram that is too large for an interface, but is not allowed be fragmented (such as broadcasts); |
| ENETUNREACH | when trying to establish a connection or send a datagram, if there is no matching entry in the routing table, or if an ICMP "destination unreachable" message is received. |
| ENOTCONN | when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected; |
| ENOBUFS | when the system runs out of memory for fragmentation buffers or other internal data structure; |
| EADDRNOTAVAIL | when an attempt is made to create a socket with a local address that matches no network interface, or when specifying an IP broadcast destination address and the network interface does not support broadcast; |

The following errors may occur when setting or getting IP options:

| | |
|---|---|
| EINVAL | An unknown socket option name was given. |
| EINVAL | The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided. |

## SEE ALSO

**connect**(2), **getsockopt**(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **icmp**(4P), **inet**(4F) **routing**(4N), **tcp**(4P), **udp**(4P),

Postel, Jon, "*Internet Protocol - DARPA Internet Program Protocol Specification*," RFC 791, Network Information Center, SRI International, Menlo Park, Calif., September 1981. (Sun 800-1063-01)

**BUGS**

Raw sockets should receive ICMP error packets relating to the protocol; currently such packets are simply discarded.

Users of higher-level protocols such as TCP and UDP should be able to see received IP options.

NAME

kb – Sun keyboard STREAMS module

CONFIG

**pseudo-device kb***number*

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/stropt.h>
#include <sundev/vuid_event.h>
#include <sundev/kbio.h>
#include <sundev/kbd.h>

ioctl(fd, I_PUSH, "kb");
```

DESCRIPTION

The **kb** STREAMS module processes byte streams generated by Sun keyboards attached to a CPU serial or parallel port. Definitions for altering keyboard translation, and reading events from the keyboard, are in **<sundev/kbio.h>** and **<sundev/kbd.h>**. *number* specifies the maximum number of keyboards supported by the system.

**kb** recognizes which keys have been typed using a set of tables for each known type of keyboard. Each translation table is an array of 128 bytes (unsigned characters). If a character value is less than 0x80, it is treated as an ASCII character (perhaps with the META bit included). Higher values indicate special characters that invoke more complicated actions.

**Keyboard Translation State**

The keyboard can be in one of the following translation modes:

| | |
|---|---|
| **TR_NONE** | Keyboard translation is turned off and up/down key codes are reported. |
| **TR_ASCII** | ASCII codes are reported. |
| **TR_EVENT** | **firm_events** (see *The SunView System Programmer's Guide — Appendix: Writing a Virtual User Input Device Driver*) are reported. |
| **TR_UNTRANS_EVENT** | **firm_events** containing unencoded keystation codes are reported for all input events within the window system. |

**Keyboard Translation-Table Entries**

All instances of the **kb** module share five translation tables used to convert raw keystation codes to event values. The tables are:

| | |
|---|---|
| Unshifted | Used when a key is depressed and no shifts are in effect. |
| Shifted | Used when a key is depressed and a Shift key is being held down. |
| Caps Lock | Used when a key is depressed and Caps Lock is in effect. |
| Controlled | Used when a key is depressed and the Control key is being held down (regardless of whether a Shift key is being held down or Caps Lock is in effect). |
| Key Up | Used when a key is released. |

Each key on the keyboard has a "key station" code which is a number from 0 to 127. This number is used as an index into the translation table that is currently in effect. If the corresponding entry in that translation table is a value from 0 to 127, this value is treated as an ASCII character, and that character is the result of the translation.

If the entry is a value from 128 to 255, it is a "special" entry. Special entry values are classified according to the value of the high-order bits. The high-order value for each class is defined as a constant, as shown in the list below. The value of the low-order bits, when added to this constant, distinguishes between keys within each class:

SHIFTKEYS 0x80 — A shift key. The value of the particular shift key is added to determine which shift mask to apply:

| | |
|---|---|
| CAPSLOCK 0 | "Caps Lock" key. |
| SHIFTLOCK 1 | "Shift Lock" key. |
| LEFTSHIFT 2 | Left-hand "Shift" key. |
| RIGHTSHIFT 3 | Right-hand "Shift" key. |
| LEFTCTRL 4 | Left-hand (or only) "Control" key. |
| RIGHTCTRL 5 | Right-hand "Control" key. |

BUCKYBITS 0x90 — Used to toggle mode-key-up/down status without altering the value of an accompanying ASCII character. (The actual bit-position value, minus 7, is added.)

| | |
|---|---|
| METABIT 0 | The "Meta" key was pressed along with the key. This is the only user-accessible bucky bit. |
| SYSTEMBIT 1 | The "System" key was pressed. This is a place holder to indicate which key is the system-abort key. |

FUNNY 0xA0 — Performs various functions depending on the value of the low 4 bits:

| | |
|---|---|
| NOP 0xA0 | Does nothing. |
| OOPS 0xA1 | Exists, but is undefined. |
| HOLE 0xA2 | There is no key in this position on the keyboard, and the position-code should not be used. |
| NOSCROLL 0xA3 | Alternately sends ^S and ^Q. |
| CTRLS 0xA4 | Sends ^S and toggles NOSCROLL key. |
| CTRLQ 0xA5 | Sends ^Q and toggles NOSCROLL key. |
| RESET 0xA6 | Keyboard reset. |
| ERROR 0xA7 | The keyboard driver detected an internal error. |
| IDLE 0xA8 | The keyboard is idle (no keys down). |
| 0xA9 — 0xAF | Reserved for nonparameterized functions. |

STRING 0xB0 — The low-order bits index a table of strings. When a key with a STRING entry is depressed, the characters in the null-terminated string for that key are sent, character by character. The maximum length is defined as:

```
KTAB_STRLEN 10
```

Individual string numbers are defined as:

```
HOMEARROW    0x00
UPARROW      0x01
DOWNARROW    0x02
LEFTARROW    0x03
RIGHTARROW   0x04
```

String numbers 0x05 — 0x0F are available for custom entries.

LEFTFUNC 0xC0
RIGHTFUNC 0xD0
TOPFUNC 0xE0
BOTTOMFUNC 0xF0 Function keys. The low 4 bits indicate the function key number within the group:

| | |
|---|---|
| LF(*n*) | (LEFTFUNC+(*n*)-1) |
| RF(*n*) | (RIGHTFUNC+(*n*)-1) |
| TF(*n*) | (TOPFUNC+(*n*)-1) |
| BF(*n*) | (BOTTOMFUNC+(*n*)-1) |

There are 64 keys reserved for function keys. The actual positions may not be on left/right/top/bottom of the keyboard, although they usually are.

In **TR_ASCII** mode, when a function key is pressed, the following escape sequence is sent:

<ESC>[0....9z

where <ESC> is a single escape character and "0...9" indicates the decimal representation of the function-key value. For example, function key R1 sends the sequence:

<ESC>[208z

because the decimal value of RF(1) is 208. In **TR_EVENT** mode, if there is a VUID event code for the function key in question, an event with that event code is generated; otherwise, individual events for the characters of the escape sequence are generated.

IOCTLS

Two **ioctl**s set and retrieve the current translation mode of a keyboard:

**KIOCTRANS** The argument is a pointer to an **int**. The translation mode is set to the value in the **int** pointed to by the argument.

**KIOCGTRANS** The argument is a pointer to an **int**. The current translation mode is stored in the **int** pointed to by the argument.

**ioctl**s for changing and retrieving entries from the keyboard translation table use the **kiockey** structure:

```
struct  kiockey {
        int     kio_tablemask;  /* Translation table (one of: 0, CAPSMASK,
                                   SHIFTMASK, CTRLMASK, UPMASK) */
#define KIOCABORT1    -1        /* Special "mask": abort1 keystation */
#define KIOCABORT2    -2        /* Special "mask": abort2 keystation */
        u_char  kio_station;    /* Physical keyboard key station (0-127) */
        u_char  kio_entry;      /* Translation table station's entry */
        char    kio_string[10]; /* Value for STRING entries (null terminated) */
};
```

**KIOCSETKEY** The argument is a pointer to a **kiockey** structure. The translation table entry referred to by the values in that structure is changed.

**kio_tablemask** specifies which of the five translation tables contains the entry to be modified:

| | |
|---|---|
| UPMASK 0x0080 | "Key Up" translation table. |
| CTRLMASK 0x0030 | "Controlled" translation table. |
| SHIFTMASK 0x000E | "Shifted" translation table. |
| CAPSMASK 0x0001 | "Caps Lock" translation table. |
| (No shift keys pressed or locked) | "Unshifted" translation table. |

**kio_station** specifies the keystation code for the entry to be modified. The value of **kio_entry** is stored in the entry in question. If **kio_entry** is between STRING and STRING+15, the string contained in **kio_string** is copied to the appropriate string table entry. This call may return **EINVAL** if there are invalid arguments.

There are a couple special values of **kio_tablemask** that affect the two step "break to the PROM monitor" sequence. The usual sequence is **SETUP–a** or **L1–a**. If **kio_tablemask** is **KIOCABORT1** then the value of **kio_station** is set to be the first keystation in the sequence. If **kio_tablemask** is **KIOCABORT2** then the value of **kio_station** is set to be the second keystation in the sequence.

**KIOCGETKEY** The argument is a pointer to a **kiockey** structure. The current value of the keyboard translation table entry specified by **kio_tablemask** and **kio_station** is stored in the structure pointed to by the argument. This call may return **EINVAL** if there are invalid arguments.

**KIOCTYPE** The argument is a pointer to an **int**. A code indicating the type of the keyboard is stored in the **int** pointed to by the argument:

| | |
|---|---|
| **KB_KLUNK** | Micro Switch 103SD32-2 |
| **KB_VT100** | Keytronics VT100 compatible |
| **KB_SUN2** | Sun-2 keyboard |
| **KB_SUN3** | Type 3 keyboard |
| **KB_SUN4** | Type 4 keyboard |
| **KB_ASCII** | ASCII terminal masquerading as keyboard |

–1 is stored in the **int** pointed to by the argument if the keyboard type is unknown.

**KIOCCMD** The argument is a pointer to an **int**. The command specifed by the value of the **int** pointed to by the argument is sent to the keyboard. The commands that can be sent are:

Commands to the Sun-2, Type 3, and Type 4 keyboard:

| | |
|---|---|
| **KBD_CMD_RESET** | Reset keyboard as if power-up. |
| **KBD_CMD_BELL** | Turn on the bell. |
| **KBD_CMD_NOBELL** | Turn off the bell |

Commands to the Type 3 and Type 4 keyboard:

| | |
|---|---|
| **KBD_CMD_CLICK** | Turn on the click annunciator. |
| **KBD_CMD_NOCLICK** | Turn off the click annunciator. |

Inappropriate commands for particular keyboard types are ignored. Since there is no reliable way to get the state of the bell or click (because we cannot query the keyboard, and also because a process could do writes to the appropriate serial driver — thus going around this **ioctl**) we do not provide an equivalent **ioctl** to query its state.

**KIOCSDIRECT**
**KIOCGDIRECT** These **ioctl**s are supported for compatibility with the system keyboard device **/dev/kbd**. **KIOCSDIRECT** has no effect, and **KIOCGDIRECT** always returns 1.

## INDEX STRUCTURES

There is a hierarchy of structures for accessing keyboard translation data. The array *keytables* contains pointers to the translation data for each of the known keyboard types:

```
struct keyboard *keytables[ ] = {
        &keyindex_ms,
        &keyindex_vt,
        &keyindex_s2,
        &keyindex_s3,
};
```

Each keyboard type is described by a **struct keyboard** that contains pointers to the five translation-tables ("Unshifted", "Shifted", "Caps Locked" "Controlled", and "Key Up") associated with that type, plus bit-masks that indicate what state can persist with no keys pressed, and the key-pair used as the abort

sequence for the system.

An array **keystringtab** contains the strings sent by various keys, and can be accessed by any translation:

```
#define kstescinit(c)  {'\033', '[', 'c', '\0'}
char keystringtab[16][KTAB_STRLEN] = {
        kstescinit(H),          /*home*/
        kstescinit(A),          /*up*/
        kstescinit(B),          /*down*/
        kstescinit(D),          /*left*/
        kstescinit(C),          /*right*/
};
```

**Index Structure for the Type 4 Keyboard**

```
/* Index to keymaps for Type 4 keyboard */
static struct keyboard keyindex_s4 = {
        &keytab_s4_lc,
        &keytab_s4_uc,
        &keytab_s4_cl,
        &keytab_s4_ct,
        &keytab_s4_up,
        0x0000,                 /* Shift bits which stay on with idle keyboard */
        0x0000,                 /* Bucky bits which stay on with idle keyboard */
        1,                      77, /* abort keys */
        CAPSMASK,               /* Shift bits which toggle on down event */
};
```

**Index Structure for the Type 3 Keyboard**

```
static struct keyboard keyindex_s3 = {
        &keytab_s3_lc,
        &keytab_s3_uc,
        &keytab_s3_cl,
        &keytab_s3_ct,
        &keytab_s3_up,
        0x0000,                 /* Shift bits that stay on with idle keyboard */
        0x0000,                 /* Bucky bits that stay on with idle keyboard */
        0x01, 0x4d,             /* Abort sequence  L1-A */
        CAPSMASK,               /* Shift bits that toggle on down event */
};
```

**Index Structure for the Sun-2 Keyboard**

```
static struct keyboard keyindex_s2 = {
        &keytab_s2_lc,
        &keytab_s2_uc,
        &keytab_s2_cl,
        &keytab_s2_ct,
        &keytab_s2_up,
        CAPSMASK,               /* Shift bits that stay on with idle keyboard */
        0x0000,                 /* Bucky bits that stay on with idle keyboard */
        0x01, 0x4d,             /* Abort sequence L1-A */
        0x0000,                 /* Shift bits that toggle on down event */
};
```

**Index Structure for the Micro Switch 103SD32-2 Keyboard**

```
static struct keyboard keyindex_ms = {
        &keytab_ms_lc,
        &keytab_ms_uc,
        &keytab_ms_cl,
        &keytab_ms_ct,
        &keytab_ms_up,
        CTLSMASK,               /* Shift bits that stay on with idle keyboard */
        0x0000,                 /* Bucky bits that stay on with idle keyboard */
        0x01, 0x4d,             /* Abort sequence L1-A */
        0x0000,                 /* Shift bits that toggle on down event */
};
```

**Index Structure for the VT100-Style Keyboard**

```
static struct keyboard keyindex_vt = {
        &keytab_vt_lc,
        &keytab_vt_uc,
        &keytab_vt_cl,
        &keytab_vt_ct,
        &keytab_vt_up,
        CAPSMASK+CTLSMASK,      /* Shift keys that stay on with idle keyboard */
        0x0000,                 /* Bucky bits that stay on with idle keyboard */
        0x01, 0x3b,             /* Abort sequence SETUP-A */
        0x0000,                 /* Shift bits that toggle on down event */
};
```

**DEFAULT TRANSLATION TABLES**

**Type 4 Keyboard**

**Unshifted**

| *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* |
|---|---|---|---|---|---|---|---|
| 00 HOLE | 01 BUCKYBITS+ SYSTEMBIT | 02 HOLE | 03 LF(2) | 04 HOLE | 05 TF(1) | 06 TF(2) | 07 TF (10) |
| 08 TF(3) | 09 TF(11) | 0A TF(4) | 0B TF(12) | 0C TF(5) | 0D HOLE | 0E TF (6) | 0F HOLE |
| 10 TF(7) | 11 TF(8) | 12 TF(9) | 13 ALT | 14 HOLE | 15 RF(1) | 16 RF(2) | 17 RF (3) |
| 18 HOLE | 19 LF(3) | 1A LF(4) | 1B HOLE | 1C HOLE | 1D c ('[') | 1E '1' | 1F '2' |
| 20 '3' | 21 '4' | 22 '5' | 23 '6' | 24 '7' | 25 '8' | 26 '9' | 27 '0' |
| 28 '-' | 29 '=' | 2A '`' | 2B '\b' | 2C HOLE | 2D RF(4) | 2E RF(5) | 2F RF (6) |
| 30 BF(13) | 31 LF(5) | 32 BF(10) | 33 LF (6) | 34 HOLE | 35 '\t' | 36 'q' | 37 'w' |
| 38 'e' | 39 'r' | 3A 't' | 3B 'y' | 3C 'u' | 3D 'i' | 3E 'o' | 3F 'p' |
| 40 '[' | 41 ']' | 42 0x7F | 43 OOPS (temp) | 44 RF(7) | 45 STRING+ UPARROW | 46 RF(9) | 47 BF (15) |
| 48 LF(7) | 49 LF (8) | 4A HOLE | 4B HOLE | 4C SHIFTKEYS+ LEFTCTRL | 4D 'a' | 4E 's' | 4F 'd' |
| 50 'f' | 51 'g' | 52 'h' | 53 'j' | 54 'k' | 55 'l' | 56 ';' | 57 '\'' |
| 58 '\\' | 59 '\r' | 5A BF(11) | 5B STRING+ LEFTARROW | 5C RF(11) | 5D STRING+ RIGHTARROW | 5E BF(8) | 5F LF (9) |
| 60 HOLE | 61 LF(10) | 62 BF (16) | 63 SHIFTKEYS+ LEFTSHIFT | 64 'z' | 65 'x' | 66 'c' | 67 'v' |
| 68 'b' | 69 'n' | 6A 'm' | 6B ',' | 6C '.' | 6D '/' | 6E SHIFTKEYS+ RIGHTSHIFT | 6F '\n' |
| 70 RF(13) | 71 STRING+ DOWNARROW | 72 RF(15) | 73 HOLE | 74 HOLE | 75 HOLE | 76 LF (16) | 77 SHIFTKEYS+ CAPSLOCK |
| 78 BUCKYBITS+ METABIT | 79 ' ' | 7A BUCKYBITS+ METABIT | 7B HOLE | 7C HOLE | 7D BF (14) | 7E ERROR | 7F IDLE |

### Type 4 Keyboard
#### Shifted

| *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* |
|---|---|---|---|---|---|---|---|
| 00 HOLE | 01 BUCKYBITS+ SYSTEMBIT | 02 HOLE | 03 LF(2) | 04 HOLE | 05 TF(1) | 06 TF(2) | 07 TF (10) |
| 08 TF(3) | 09 TF(11) | 0A TF(4) | 0B TF(12) | 0C TF(5) | 0D HOLE | 0E TF (6) | 0F HOLE |
| 10 TF(7) | 11 TF(8) | 12 TF(9) | 13 ALT | 14 HOLE | 15 RF(1) | 16 RF(2) | 17 RF (3) |
| 18 HOLE | 19 LF(3) | 1A LF(4) | 1B HOLE | 1C HOLE | 1D c ('[') | 1E '!' | 1F '@' |
| 20 '#' | 21 '$' | 22 '%' | 23 '^' | 24 '&' | 25 '*' | 26 '(' | 27 ')' |
| 28 '_' | 29 '+' | 2A '~' | 2B '\b' | 2C HOLE | 2D RF(4) | 2E RF(5) | 2F RF (6) |
| 30 BF(13) | 31 LF(5) | 32 BF(10) | 33 LF (6) | 34 HOLE | 35 '\t' | 36 'Q' | 37 'W' |
| 38 'E' | 39 'R' | 3A 'T' | 3B 'Y' | 3C 'U' | 3D 'I' | 3E 'O' | 3F 'P' |
| 40 '{' | 41 '}' | 42 0x7F | 43 OOPS (temp) | 44 RF(7) | 45 STRING+ UPARROW | 46 RF(9) | 47 BF (15) |
| 48 LF(7) | 49 LF (8) | 4A HOLE | 4B HOLE | 4C SHIFTKEYS+ LEFTCTRL | 4D 'A' | 4E 'S' | 4F 'D' |
| 50 'F' | 51 'G' | 52 'H' | 53 'J' | 54 'K' | 55 'L' | 56 ':' | 57 '"' |
| 58 '\|' | 59 '\r' | 5A BF(11) | 5B STRING+ LEFTARROW | 5C RF(11) | 5D STRING+ RIGHTARROW | 5E BF(8) | 5F LF (9) |
| 60 HOLE | 61 LF(10) | 62 BF (16) | 63 SHIFTKEYS+ LEFTSHIFT | 64 'Z' | 65 'X' | 66 'C' | 67 'V' |
| 68 'B' | 69 'N' | 6A 'M' | 6B '<' | 6C '>' | 6D '?' | 6E SHIFTKEYS+ RIGHTSHIFT | 6F '\n' |
| 70 RF(13) | 71 STRING+ DOWNARROW | 72 RF(15) | 73 HOLE | 74 HOLE | 75 HOLE | 76 LF (16) | 77 SHIFTKEYS+ CAPSLOCK |
| 78 BUCKYBITS+ METABIT | 79 ' ' | 7A BUCKYBITS+ METABIT | 7B HOLE | 7C HOLE | 7D BF (14) | 7E ERROR | 7F IDLE |

### Type 4 Keyboard
#### Caps Locked

| *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* | *Key Value* |
|---|---|---|---|---|---|---|---|
| 00 HOLE | 01 BUCKYBITS+ SYSTEMBIT | 02 HOLE | 03 LF(2) | 04 HOLE | 05 TF(1) | 06 TF(2) | 07 TF (10) |
| 08 TF(3) | 09 TF(11) | 0A TF(4) | 0B TF(12) | 0C TF(5) | 0D HOLE | 0E TF (6) | 0F HOLE |
| 10 TF(7) | 11 TF(8) | 12 TF(9) | 13 ALT | 14 HOLE | 15 RF(1) | 16 RF(2) | 17 RF (3) |
| 18 HOLE | 19 LF(3) | 1A LF(4) | 1B HOLE | 1C HOLE | 1D c ('[') | 1E '1' | 1F '2' |
| 21 '3' | 21 '4' | 22 '5' | 23 '6' | 24 '7' | 25 '8' | 26 '9' | 27 '0' |
| 28 '-' | 29 '=' | 2A '`' | 2B '\b' | 2C HOLE | 2D RF(4) | 2E RF(5) | 2F RF (6) |
| 30 BF(23) | 32 LF(5) | 32 BF(20) | 33 LF (6) | 34 HOLE | 35 '\t' | 36 'Q' | 37 'W' |
| 38 'E' | 39 'R' | 3A 'T' | 3B 'Y' | 3C 'U' | 3D 'I' | 3E 'O' | 3F 'P' |
| 40 '[' | 41 ']' | 43 0x7F | 43 OOPS (temp) | 44 RF(7) | 45 STRING+ UPARROW | 46 RF(9) | 47 BF (15) |
| 48 LF(7) | 49 LF (8) | 4A HOLE | 4B HOLE | 4C SHIFTKEYS+ LEFTCTRL | 4D 'A' | 4E 'S' | 4F 'D' |
| 50 'F' | 51 'G' | 52 'H' | 53 'J' | 54 'K' | 55 'L' | 56 ';' | 57 '\'' |
| 58 '\\' | 59 '\r' | 5A BF(11) | 5B STRING+ LEFTARROW | 5C RF(11) | 5D STRING+ RIGHTARRROW | 5E BF(8) | 5F LF (9) |
| 60 HOLE | 61 LF(10) | 62 BF (16) | 63 SHIFTKEYS+ LEFTSHIFT | 64 'Z' | 65 'X' | 66 'C' | 67 'V' |
| 68 'B' | 69 'N' | 6A 'M' | 6B ',' | 6C '.' | 6D '/' | 6E SHIFTKEYS+ RIGHTSHIFT | 6F '\n' |
| 70 RF(13) | 71 STRING+ DOWNARROW | 72 RF(15) | 73 HOLE | 74 HOLE | 75 HOLE | 76 LF (16) | 77 SHIFTKEYS+ CAPSLOCK |
| 78 BUCKYBITS+ METABIT | 79 ' ' | 7A BUCKYBITS+ METABIT | 7B HOLE | 7C HOLE | 7D BF (14) | 7E ERROR | 7F IDLE |

### Type 4 Keyboard

#### Controlled

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF(10) |
| 08 | TF(3) | 09 | TF(11) | 0A | TF(4) | 0B | TF(12) | 0C | TF(5) | 0D | HOLE | 0E | TF(6) | 0F | HOLE |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | ALT | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF(3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | HOLE | 1C | HOLE | 1D | c('[') | 1E | '1' | 1F | c('@') |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | c('^') | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | c('_') | 29 | '=' | 2A | c('~') | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF(6) |
| 30 | BF(13) | 31 | LF(5) | 32 | BF(10) | 33 | LF(6) | 34 | HOLE | 35 | '\t' | 36 | c('q') | 37 | c('w') |
| 38 | c('e') | 39 | c('r') | 3A | c('t') | 3B | c('y') | 3C | c('u') | 3D | c('i') | 3E | c('o') | 3F | c('p') |
| 40 | c('[') | 41 | c(']') | 42 | 0x7F | 43 | OOPS (temp) | 44 | RF(7) | 45 | STRING+ UPARROW | 46 | RF(9) | 47 | BF(15) |
| 48 | LF(7) | 49 | LF(8) | 4A | HOLE | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | c('a') | 4E | c('s') | 4F | c('d') |
| 50 | c('f') | 51 | c('g') | 52 | c('h') | 53 | c('j') | 54 | c('k') | 55 | c('l') | 56 | ';' | 57 | '\'' |
| 58 | c('\\') | 59 | '\r' | 5A | BF(11) | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | BF(8) | 5F | LF(9) |
| 60 | HOLE | 61 | LF(10) | 62 | BF(16) | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | c('z') | 65 | c('x') | 66 | c('c') | 67 | c('v') |
| 68 | c('b') | 69 | c('n') | 6A | c('m') | 6B | ',' | 6C | '.' | 6D | c('_') | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF(15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | LF(16) | 77 | SHIFTKEYS+ CAPSLOCK |
| 78 | BUCKYBITS+ METABIT | 79 | c(' ') | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | BF(14) | 7E | ERROR | 7F | IDLE |

### Type 4 Keyboard

#### Key Up

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | OOPS | 04 | HOLE | 05 | OOPS | 06 | OOPS | 07 | HOLE |
| 08 | OOPS | 09 | HOLE | 0A | OOPS | 0B | HOLE | 0C | OOPS | 0D | HOLE | 0E | OOPS | 0F | HOLE |
| 10 | OOPS | 11 | OOPS | 12 | OOPS | 13 | OOPS | 14 | HOLE | 15 | OOPS | 16 | OOPS | 17 | NOP |
| 18 | HOLE | 19 | OOPS | 1A | OOPS | 1B | HOLE | 1C | HOLE | 1D | NOP | 1E | NOP | 1F | NOP |
| 20 | NOP | 21 | NOP | 22 | NOP | 23 | NOP | 24 | NOP | 25 | NOP | 26 | NOP | 27 | NOP |
| 28 | NOP | 29 | NOP | 2A | NOP | 2B | NOP | 2C | HOLE | 2D | OOPS | 2E | OOPS | 2F | NOP |
| 30 | HOLE | 31 | OOPS | 32 | HOLE | 33 | OOPS | 34 | HOLE | 35 | NOP | 36 | NOP | 37 | NOP |
| 38 | NOP | 39 | NOP | 3A | NOP | 3B | NOP | 3C | NOP | 3D | NOP | 3E | NOP | 3F | NOP |
| 40 | NOP | 41 | NOP | 42 | NOP | 43 | HOLE | 44 | OOPS | 45 | OOPS | 46 | NOP | 47 | HOLE |
| 48 | OOPS | 49 | OOPS | 4A | HOLE | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | NOP | 4E | NOP | 4F | NOP |
| 50 | NOP | 51 | NOP | 52 | NOP | 53 | NOP | 54 | NOP | 55 | NOP | 56 | NOP | 57 | NOP |
| 58 | NOP | 59 | NOP | 5A | HOLE | 5B | OOPS | 5C | OOPS | 5D | NOP | 5E | HOLE | 5F | OOPS |
| 60 | OOPS | 61 | OOPS | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | NOP | 65 | NOP | 66 | NOP | 67 | NOP |
| 68 | NOP | 69 | NOP | 6A | NOP | 6B | NOP | 6C | NOP | 6D | NOP | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | NOP |
| 70 | OOPS | 71 | OOPS | 72 | NOP | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | NOP |
| 78 | BUCKYBITS+ METABIT | 79 | NOP | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | RESET |

## Type 3 Keyboard

### Unshifted

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF (2) | 07 | HOLE |
| 08 | TF(3) | 09 | HOLE | 0A | TF(4) | 0B | HOLE | 0C | TF(5) | 0D | HOLE | 0E | TF (6) | 0F | HOLE |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | ALT | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF (3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | HOLE | 1C | HOLE | 1D | c ('[') | 1E | '1' | 1F | '2' |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | '6' | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | '–' | 29 | '=' | 2A | '`' | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF (6) |
| 30 | HOLE | 31 | LF(5) | 32 | HOLE | 33 | LF (6) | 34 | HOLE | 35 | '\t' | 36 | 'q' | 37 | 'w' |
| 38 | 'e' | 39 | 'r' | 3A | 't' | 3B | 'y' | 3C | 'u' | 3D | 'i' | 3E | 'o' | 3F | 'p' |
| 40 | '[' | 41 | ']' | 42 | 0x7F | 43 | HOLE | 45 | RF(7) | 45 | STRING+ UPARROW | 46 | RF (9) | 47 | HOLE |
| 48 | LF(7) | 49 | LF(8) | 4A | LF (40) | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | 'a' | 4E | 's' | 4F | 'd' |
| 50 | 'f' | 51 | 'g' | 52 | 'h' | 53 | 'j' | 54 | 'k' | 55 | 'l' | 56 | ';' | 57 | '\'' |
| 58 | '\' | 59 | '\r' | 5A | HOLE | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | HOLE | 5F | LF (9) |
| 60 | LF(15) | 61 | LF (10) | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | 'z' | 65 | 'x' | 66 | 'c' | 67 | 'v' |
| 68 | 'b' | 69 | 'n' | 6A | 'm' | 6B | ',' | 6C | '.' | 6D | '/' | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF (15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | SHIFTKEYS+ CAPSLOCK |
| 78 | BUCKYBITS+ METABIT | 79 | ' ' | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | ERROR | 7F | IDLE |

### Shifted

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF (2) | 07 | HOLE |
| 08 | TF(3) | 09 | HOLE | 0A | TF(4) | 0B | HOLE | 0C | TF(5) | 0D | HOLE | 0E | TF (6) | 0F | HOLE |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | ALT | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF (3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | HOLE | 1C | HOLE | 1D | c ('[') | 1E | '!' | 1F | '@' |
| 20 | '#' | 21 | '$' | 22 | '%' | 23 | '^' | 24 | '&' | 25 | '*' | 26 | '(' | 27 | ')' |
| 28 | '_' | 29 | '+' | 2A | '~' | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF (6) |
| 30 | HOLE | 31 | LF(5) | 32 | HOLE | 33 | LF (6) | 34 | HOLE | 35 | '\t' | 36 | 'Q' | 37 | 'W' |
| 38 | 'E' | 39 | 'R' | 3A | 'T' | 3B | 'Y' | 3C | 'U' | 3D | 'I' | 3E | 'O' | 3F | 'P' |
| 40 | '{' | 41 | '}' | 42 | 0x7F | 43 | HOLE | 44 | RF(7) | 45 | STRING+ UPARROW | 46 | RF (9) | 47 | HOLE |
| 48 | LF(7) | 49 | LF (8) | 4A | HOLE | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | 'A' | 4E | 'S' | 4F | 'D' |
| 50 | 'F' | 51 | 'G' | 52 | 'H' | 53 | 'J' | 54 | 'K' | 55 | 'L' | 56 | ':' | 57 | '"' |
| 58 | '|' | 59 | '\r' | 5A | HOLE | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | HOLE | 5F | LF (9) |
| 60 | LF(15) | 61 | LF (10) | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | 'Z' | 65 | 'X' | 66 | 'C' | 67 | 'V' |
| 68 | 'B' | 69 | 'N' | 6A | 'M' | 6B | '<' | 6C | '>' | 6D | '?' | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF (15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | SHIFTKEYS+ CAPSLOCK |
| 78 | BUCKYBITS+ METABIT | 79 | ' ' | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | ERROR | 7F | IDLE |

## Type 3 Keyboard

### Caps Locked

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF (2) | 07 | HOLE |
| 08 | TF(3) | 09 | HOLE | 0A | TF(4) | 0B | HOLE | 0C | TF(5) | 0D | HOLE | 0E | TF (6) | 0F | HOLE |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | ALT | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF (3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | HOLE | 1C | HOLE | 1D | c ('[') | 1E | '1' | 1F | '2' |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | '6' | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | '-' | 29 | '=' | 2A | '`' | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF (6) |
| 30 | HOLE | 31 | LF(5) | 32 | HOLE | 33 | LF (6) | 34 | HOLE | 35 | '\t' | 36 | 'Q' | 37 | 'W' |
| 38 | 'E' | 39 | 'R' | 3A | 'T' | 3B | 'Y' | 3C | 'U' | 3D | 'I' | 3E | 'O' | 3F | 'P' |
| 40 | '[' | 41 | ']' | 42 | 0x7F | 43 | HOLE | 44 | RF(7) | 45 | STRING+ UPARROW | 46 | RF (9) | 47 | HOLE |
| 48 | LF(7) | 49 | LF (8) | 4A | HOLE | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | 'A' | 4E | 'S' | 4F | 'D' |
| 50 | 'F' | 51 | 'G' | 52 | 'H' | 53 | 'J' | 54 | 'K' | 55 | 'L' | 56 | ';' | 57 | '\'' |
| 58 | '\' | 59 | '\r' | 5A | HOLE | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | HOLE | 5F | LF (9) |
| 60 | LF(15) | 61 | LF (10) | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | 'Z' | 65 | 'X' | 66 | 'C' | 67 | 'V' |
| 68 | 'B' | 69 | 'N' | 6A | 'M' | 6B | ',' | 6C | '.' | 6D | '/' | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF (15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | SHIFTKEYS+ CAPSLOCK |
| 78 | BUCKYBITS+ METABIT | 79 | ' ' | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | ERROR | 7F | IDLE |

### Controlled

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF (2) | 07 | HOLE |
| 08 | TF(3) | 09 | HOLE | 0A | TF(4) | 0B | HOLE | 0C | TF(5) | 0D | HOLE | 0E | TF (6) | 0F | HOLE |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | ALT | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF (3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | HOLE | 1C | HOLE | 1D | c('[') | 1E | '1' | 1F | c ('@') |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | c ('^') | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | c('_') | 29 | '=' | 2A | c('^') | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF (6) |
| 30 | HOLE | 31 | LF(5) | 32 | HOLE | 33 | LF(6) | 34 | HOLE | 35 | '\t' | 36 | c('q') | 37 | c ('w') |
| 38 | c('e') | 39 | c('r') | 3A | c('t') | 3B | c('y') | 3C | c('u') | 3D | c('i') | 3E | c('o') | 3F | c ('p') |
| 40 | c('[') | 41 | c(']') | 42 | 0x7F | 43 | HOLE | 44 | RF(7) | 45 | STRING+ UPARROW | 46 | RF (9) | 47 | HOLE |
| 48 | LF(7) | 49 | LF(8) | 4A | HOLE | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | c('a') | 4E | c('s') | 4F | c ('d') |
| 50 | c('f') | 51 | c('g') | 52 | c('h') | 53 | c('j') | 54 | c('k') | 55 | c ('l') | 56 | ';' | 57 | '\'' |
| 58 | c('\') | 59 | '\r' | 5A | HOLE | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | HOLE | 5F | LF (9) |
| 60 | LF(15) | 61 | LF(10) | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | c('z') | 65 | c('x') | 66 | c('c') | 67 | c ('v') |
| 68 | c('b') | 69 | c('n') | 6A | c('m') | 6B | ',' | 6C | '.' | 6D | c ('_') | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF (15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | SHIFTKEYS+ CAPSLOCK |
| 78 | BUCKYBITS+ METABIT | 79 | c (' ') | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | ERROR | 7F | IDLE |

## Type 3 Keyboard

### Key Up

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | OOPS | 04 | HOLE | 05 | OOPS | 06 | OOPS | 07 | HOLE |
| 08 | OOPS | 09 | HOLE | 0A | OOPS | 0B | HOLE | 0C | OOPS | 0D | HOLE | 0E | OOPS | 0F | HOLE |
| 10 | OOPS | 11 | OOPS | 12 | OOPS | 13 | OOPS | 14 | HOLE | 15 | OOPS | 16 | OOPS | 17 | NOP |
| 18 | HOLE | 19 | OOPS | 1A | OOPS | 1B | HOLE | 1C | HOLE | 1D | NOP | 1E | NOP | 1F | NOP |
| 20 | NOP | 21 | NOP | 22 | NOP | 23 | NOP | 24 | NOP | 25 | NOP | 26 | NOP | 27 | NOP |
| 28 | NOP | 29 | NOP | 2A | NOP | 2B | NOP | 2C | HOLE | 2D | OOPS | 2E | OOPS | 2F | NOP |
| 30 | HOLE | 31 | OOPS | 32 | HOLE | 33 | OOPS | 34 | HOLE | 35 | NOP | 36 | NOP | 37 | NOP |
| 38 | NOP | 39 | NOP | 3A | NOP | 3B | NOP | 3C | NOP | 3D | NOP | 3E | NOP | 3F | NOP |
| 40 | NOP | 41 | NOP | 42 | NOP | 43 | HOLE | 44 | OOPS | 45 | OOPS | 46 | NOP | 47 | HOLE |
| 48 | OOPS | 49 | OOPS | 4A | HOLE | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | NOP | 4E | NOP | 4F | NOP |
| 50 | NOP | 51 | NOP | 52 | NOP | 53 | NOP | 54 | NOP | 55 | NOP | 56 | NOP | 57 | NOP |
| 58 | NOP | 59 | NOP | 5A | HOLE | 5B | OOPS | 5C | OOPS | 5D | NOP | 5E | HOLE | 5F | OOPS |
| 60 | OOPS | 61 | OOPS | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | NOP | 65 | NOP | 66 | NOP | 67 | NOP |
| 68 | NOP | 69 | NOP | 6A | NOP | 6B | NOP | 6C | NOP | 6D | NOP | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | NOP |
| 70 | OOPS | 71 | OOPS | 72 | NOP | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | NOP |
| 78 | BUCKYBITS+ METABIT | 79 | NOP | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | RESET |

## Sun-2 Keyboard

### Unshifted

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | LF(11) | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF (11) |
| 08 | TF (3) | 09 | TF(12) | 0A | TF(4) | 0B | TF(13) | 0C | TF(5) | 0D | TF(14) | 0E | TF(6) | 0F | TF(15) |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | TF(10) | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF (3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | LF(12) | 1C | HOLE | 1D | c('[') | 1E | '1' | 1F | '2' |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | '6' | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | '–' | 29 | '=' | 2A | '`' | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF (6) |
| 30 | HOLE | 31 | LF(5) | 32 | LF(13) | 33 | LF (6) | 34 | HOLE | 35 | '\t' | 36 | 'q' | 37 | 'w' |
| 38 | 'e' | 39 | 'r' | 3A | 't' | 3B | 'y' | 3C | 'u' | 3D | 'i' | 3E | 'o' | 3F | 'p' |
| 40 | '[' | 41 | ']' | 42 | 0x7F | 43 | HOLE | 44 | RF(7) | 45 | STRING+ UPARROW | 46 | RF (9) | 47 | HOLE |
| 48 | LF(7) | 49 | LF(8) | 4A | LF (14) | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | 'a' | 4E | 's' | 4F | 'd' |
| 50 | 'f' | 51 | 'g' | 52 | 'h' | 53 | 'j' | 54 | 'k' | 55 | 'l' | 56 | ';' | 57 | '\'' |
| 58 | '\' | 59 | '\r' | 5A | HOLE | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | HOLE | 5F | LF (9) |
| 60 | LF(15) | 61 | LF (10) | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | 'z' | 65 | 'x' | 66 | 'c' | 67 | 'v' |
| 68 | 'b' | 69 | 'n' | 6A | 'm' | 6B | ',' | 6C | '.' | 6D | '/' | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF (15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 70 | BUCKYBITS+ METABIT | 71 | ' ' | 72 | BUCKYBITS+ METABIT | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | ERROR | 77 | IDLE |

## Sun-2 Keyboard

### Shifted

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | LF(11) | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF (11) |
| 08 | TF (3) | 00 | TF(12) | 0A | TF(4) | 0B | TF(13) | 0C | TF(5) | 0D | TF(14) | 0E | TF(6) | 0F | TF(15) |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | TF(10) | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF (3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | LF(12) | 1C | HOLE | 1D | c('[') | 1E | '!' | 1F | '@' |
| 20 | '#' | 21 | '$' | 22 | '%' | 23 | '^' | 24 | '&' | 25 | '*' | 26 | '(' | 27 | ')' |
| 28 | '_' | 29 | '+' | 2A | '~' | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF (6) |
| 30 | HOLE | 31 | LF(5) | 32 | LF(13) | 33 | LF (6) | 34 | HOLE | 35 | '\t' | 36 | 'Q' | 37 | 'W' |
| 38 | 'E' | 39 | 'R' | 3A | 'T' | 3B | 'Y' | 3C | 'U' | 3D | 'I' | 3E | 'O' | 3F | 'P' |
| 40 | '{' | 41 | '}' | 42 | 0x7F | 43 | HOLE | 44 | RF(7) | 45 | STRING+ UPARROW | 46 | RF (9) | 47 | HOLE |
| 48 | LF(7) | 49 | LF(8) | 4A | LF (14) | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | 'A' | 4E | 'S' | 4F | 'D' |
| 50 | 'F' | 51 | 'G' | 52 | 'H' | 53 | 'J' | 54 | 'K' | 55 | 'L' | 56 | ':' | 57 | '"' |
| 58 | '\|' | 59 | '\r' | 5A | HOLE | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | HOLE | 5F | LF (9) |
| 60 | LF(15) | 61 | LF (10) | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | 'Z' | 65 | 'X' | 66 | 'C' | 67 | 'V' |
| 68 | 'B' | 69 | 'N' | 6A | 'M' | 6B | '<' | 6C | '>' | 6D | '?' | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF (15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | BUCKYBITS+ METABIT | 79 | ' ' | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | ERROR | 7F | IDLE |

### Caps Locked

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | LF(11) | 03 | LF(2) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF (11) |
| 08 | TF (3) | 09 | TF(12) | 0A | TF(4) | 0B | TF(13) | 0C | TF(5) | 0D | TF(14) | 0E | TF(6) | 0F | TF(15) |
| 10 | TF(7) | 11 | TF(8) | 12 | TF(9) | 13 | TF(10) | 14 | HOLE | 15 | RF(1) | 16 | RF(2) | 17 | RF (3) |
| 18 | HOLE | 19 | LF(3) | 1A | LF(4) | 1B | LF(12) | 1C | HOLE | 1D | c('[') | 1E | '1' | 1F | '2' |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | '6' | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | '-' | 29 | '=' | 2A | '`' | 2B | '\b' | 2C | HOLE | 2D | RF(4) | 2E | RF(5) | 2F | RF (6) |
| 30 | HOLE | 31 | LF(5) | 32 | LF(13) | 33 | LF (6) | 34 | HOLE | 35 | '\t' | 36 | 'Q' | 37 | 'W' |
| 38 | 'E' | 39 | 'R' | 3A | 'T' | 3B | 'Y' | 3C | 'U' | 3D | 'I' | 3E | 'O' | 3F | 'P' |
| 40 | '[' | 41 | ']' | 42 | 0x7F | 43 | HOLE | 44 | RF(7) | 45 | STRING+ UPARROW | 46 | RF (9) | 47 | HOLE |
| 48 | LF(7) | 49 | LF(8) | 4A | LF (14) | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | 'A' | 4E | 'S' | 4F | 'D' |
| 50 | 'F' | 51 | 'G' | 52 | 'H' | 53 | 'J' | 54 | 'K' | 55 | 'L' | 56 | ';' | 57 | '\'' |
| 58 | '\' | 59 | '\r' | 5A | HOLE | 5B | STRING+ LEFTARROW | 5C | RF(11) | 5D | STRING+ RIGHTARROW | 5E | HOLE | 5F | LF (9) |
| 60 | LF(15) | 61 | LF (10) | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | 'Z' | 65 | 'X' | 66 | 'C' | 67 | 'V' |
| 68 | 'B' | 69 | 'N' | 6A | 'M' | 6B | ',' | 6C | '.' | 6D | '/' | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | '\n' |
| 70 | RF(13) | 71 | STRING+ DOWNARROW | 72 | RF (15) | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | BUCKYBITS+ METABIT | 79 | ' ' | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | ERROR | 7F | IDLE |

## Sun-2 Keyboard

### Controlled

| *KeyValue* | *KeyValue* | *KeyValue* | *KeyValue* | *Key* | *Value* | *KeyValue* | *Key* | *Value* | *KeyValue* |
|---|---|---|---|---|---|---|---|---|---|
| 00 HOLE | 01 BUCKYBITS+ | 02 LF(11) | 03 LF(2) | 04 | HOLE | 05 TF(1) | 06 | TF(2) | 07 TF (11) |
| 08 TF(3) | 09 TF(12) | 0A TF(4) | 0B TF(13) | 0C | TF(5) | 0D TF(14) | 0E | TF(6) | 0F TF (15) |
| 10 TF(7) | 11 TF(8) | 12 TF(9) | 13 TF(10) | 14 | HOLE | 15 RF(1) | 16 | RF(2) | 17 RF (3) |
| 18 HOLE | 19 LF(3) | 1A LF(4) | 1B LF(12) | 1C | HOLE | 1D c('[') | 1E | '1' | 1F c('@') |
| 20 '3' | 21 '4' | 22 '5' | 23 c ('^') | 24 | '7' | 25 '8' | 26 | '9' | 27 '0' |
| 28 c('_') | 29 '=' | 2A c('^') | 2B '\b' | 2C | HOLE | 2D RF(4) | 2E | RF(5) | 2F RF (6) |
| 30 HOLE | 31 LF(5) | 32 LF(13) | 33 LF(6) | 34 | HOLE | 35 '\t' | 36 | c('q') | 37 c ('w') |
| 38 c('e') | 39 c('r') | 3A c('t') | 3B c('y') | 3C | c('u') | 3D c('i') | 3E | c('o') | 3F c ('p') |
| 40 c('[') | 41 c(']') | 42 0x7F | 43 HOLE | 44 | RF(7) | 45 STRING+ UPARROW | 46 | RF (9) | 47 HOLE |
| 48 LF(7) | 49 LF(8) | 4A LF(14) | 4B HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D c('a') | 4E | c('s') | 4F c ('d') |
| 50 c('f') | 51 c('g') | 52 c('h') | 53 c('j') | 54 | c('k') | 55 c ('l') | 56 | ';' | 57 '\'' |
| 58 c('\') | 59 '\r' | 5A HOLE | 5B STRING+ LEFTARROW | 5C | RF(11) | 5D STRING+ RIGHTARROW | 5E | HOLE | 5F LF (9) |
| 60 LF(15) | 61 LF(10) | 62 HOLE | 63 SHIFTKEYS+ LEFTSHIFT | 64 | c('z') | 65 c('x') | 66 | c('c') | 67 c ('v') |
| 68 c('b') | 69 c('n') | 6A c('m') | 6B ',' | 6C | '.' | 6D c ('_') | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F '\n' |
| 70 RF(13) | 71 STRING+ DOWNARROW | 72 RF (15) | 73 HOLE | 74 | HOLE | 75 HOLE | 76 | HOLE | 77 HOLE |
| 78 BUCKYBITS+ METABIT | 79 c (' ') | 7A BUCKYBITS+ METABIT | 7B HOLE | 7C | HOLE | 7D HOLE | 7E | ERROR | 7F IDLE |

### Key Up

| *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | OOPS | 03 | OOPS | 04 | HOLE | 05 | OOPS | 06 | OOPS | 07 | OOPS |
| 08 | OOPS | 09 | OOPS | 0A | OOPS | 0B | OOPS | 0C | OOPS | 0D | OOPS | 0E | OOPS | 0F | OOPS |
| 10 | OOPS | 11 | OOPS | 12 | OOPS | 13 | OOPS | 14 | HOLE | 15 | OOPS | 16 | OOPS | 17 | NOP |
| 18 | HOLE | 19 | OOPS | 1A | OOPS | 1B | OOPS | 1C | HOLE | 1D | NOP | 1E | NOP | 1F | NOP |
| 20 | NOP | 21 | NOP | 22 | NOP | 23 | NOP | 24 | NOP | 25 | NOP | 26 | NOP | 27 | NOP |
| 28 | NOP | 29 | NOP | 2A | NOP | 2B | NOP | 2C | HOLE | 2D | OOPS | 2E | OOPS | 2F | NOP |
| 30 | HOLE | 31 | OOPS | 32 | OOPS | 33 | OOPS | 34 | HOLE | 35 | NOP | 36 | NOP | 37 | NOP |
| 38 | NOP | 39 | NOP | 3A | NOP | 3B | NOP | 3C | NOP | 3D | NOP | 3E | NOP | 3F | NOP |
| 40 | NOP | 41 | NOP | 42 | NOP | 43 | HOLE | 44 | OOPS | 45 | OOPS | 46 | NOP | 47 | HOLE |
| 48 | OOPS | 49 | OOPS | 4A | OOPS | 4B | HOLE | 4C | SHIFTKEYS+ LEFTCTRL | 4D | NOP | 4e | NOP | 4F | NOP |
| 50 | NOP | 51 | NOP | 52 | NOP | 53 | NOP | 54 | NOP | 55 | NOP | 56 | NOP | 57 | NOP |
| 58 | NOP | 59 | NOP | 5A | HOLE | 5B | OOPS | 5C | OOPS | 5D | NOP | 5E | HOLE | 5F | OOPS |
| 60 | OOPS | 61 | OOPS | 62 | HOLE | 63 | SHIFTKEYS+ LEFTSHIFT | 64 | NOP | 65 | NOP | 66 | NOP | 67 | NOP |
| 68 | NOP | 69 | NOP | 6A | NOP | 6B | NOP | 6C | NOP | 6D | NOP | 6E | SHIFTKEYS+ RIGHTSHIFT | 6F | NOP |
| 70 | OOPS | 71 | OOPS | 72 | NOP | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | BUCKYBITS+ METABIT | 79 | NOP | 7A | BUCKYBITS+ METABIT | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | RESET |

### Micro Switch 103SD32-2 Keyboard

#### Unshifted

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | LF(2) | 03 | LF(3) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF (3) |
| 08 | TF(4) | 09 | TF(5) | 0A | TF(6) | 0B | TF(7) | 0C | TF(8) | 0D | TF(9) | 0E | TF(10) | 0F | TF (11) |
| 10 | TF(12) | 11 | TF(13) | 12 | TF(14) | 13 | c('[') | 14 | HOLE | 15 | RF (1) | 16 | '+' | 17 | '–' |
| 18 | HOLE | 19 | LF(4) | 1A | '\f' | 1B | LF (6) | 1C | HOLE | 1D | SHIFTKEYS+ CAPSLOCK | 1E | '1' | 1F | '2' |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | '6' | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | '–' | 29 | '~' | 2A | '`' | 2B | '\b' | 2C | HOLE | 2D | '7' | 2E | '8' | 2F | '9' |
| 30 | HOLE | 31 | LF(7) | 32 | STRING+ UPARROW | 33 | LF (9) | 34 | HOLE | 35 | '\t' | 36 | 'q' | 37 | 'w' |
| 38 | 'e' | 39 | 'r' | 3A | 't' | 3B | 'y' | 3C | 'u' | 3D | 'i' | 3E | 'o' | 3F | 'p' |
| 40 | '{' | 41 | '}' | 42 | '_' | 43 | HOLE | 44 | '4' | 45 | '5' | 46 | '6' | 47 | HOLE |
| 48 | STRING+ LEFTARROW | 49 | STRING+ HOMEARROW | 4A | STRING+ RIGHTARROW | 4B | HOLE | 4C | SHIFTKEYS+ SHIFTLOCK | 4D | 'a' | 4E | 's' | 4F | 'd' |
| 50 | 'f' | 51 | 'g' | 52 | 'h' | 53 | 'j' | 54 | 'k' | 55 | 'l' | 56 | ';' | 57 | ':' |
| 58 | '\|' | 59 | '\r' | 5A | HOLE | 5B | '1' | 5C | '2' | 5D | '3' | 5E | HOLE | 5F | NOSCROLL |
| 60 | STRING+ DOWNARROW | 61 | LF (15) | 62 | HOLE | 63 | HOLE | 64 | SHIFTKEYS+ LEFTSHIFT | 65 | 'z' | 66 | 'x' | 67 | 'c' |
| 68 | 'v' | 69 | 'b' | 6A | 'n' | 6B | 'm' | 6C | ',' | 6D | '.' | 6E | '/' | 6F | SHIFTKEYS+ RIGHTSHIFT |
| 70 | NOP | 71 | 0x7F | 72 | '0' | 73 | NOP | 74 | '.' | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | SHIFTKEYS+ LEFTCTRL | 7B | ' ' | 7C | SHIFTKEYS+ RIGHTCTRL | 7D | HOLE | 7E | HOLE | 7F | IDLE |

#### Shifted

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS SYSTEMBIT | 02 | LF(2) | 03 | LF(3) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF (3) |
| 08 | TF(4) | 09 | TF(5) | 0A | TF(6) | 0B | TF(7) | 0C | TF(8) | 0D | TF(9) | 0E | TF(10) | 0F | TF (11) |
| 10 | TF(12) | 11 | TF(13) | 12 | TF(14) | 13 | c('[') | 14 | HOLE | 15 | RF (1) | 16 | '+' | 17 | '–' |
| 18 | HOLE | 19 | LF(4) | 1A | '\f' | 1B | LF (6) | 1C | HOLE | 1D | SHIFTKEYS+ CAPSLOCK | 1E | '!' | 1F | '"' |
| 20 | '#' | 21 | '$' | 22 | '%' | 23 | '&' | 24 | '\'' | 25 | '(' | 26 | ')' | 27 | '0' |
| 28 | '=' | 29 | '^' | 2A | '@' | 2B | '\b' | 2C | HOLE | 2D | '7' | 2E | '8' | 2F | '9' |
| 30 | HOLE | 31 | LF(7) | 32 | STRING+ UPARROW | 33 | LF (9) | 34 | HOLE | 35 | '\t' | 36 | 'Q' | 37 | 'W' |
| 38 | 'E' | 39 | 'R' | 3A | 'T' | 3B | 'Y' | 3C | 'U' | 3D | 'I' | 3E | 'O' | 3F | 'P' |
| 40 | '[' | 41 | ']' | 42 | '_' | 43 | HOLE | 44 | '4' | 45 | '5' | 46 | '6' | 47 | HOLE |
| 48 | STRING+ LEFTARROW | 49 | STRING+ HOMEARROW | 4A | STRING+ RIGHTARROW | 4B | HOLE | 4C | SHIFTKEYS+ SHIFTLOCK | 4D | 'A' | 4E | 'S' | 4F | 'D' |
| 50 | 'F' | 51 | 'G' | 52 | 'H' | 53 | 'J' | 54 | 'K' | 55 | 'L' | 56 | '+' | 57 | '*' |
| 58 | '\' | 59 | '\r' | 5A | HOLE | 5B | '1' | 5C | '2' | 5D | '3' | 5E | HOLE | 5F | NOSCROLL |
| 60 | STRING+ DOWNARROW | 61 | LF (15) | 62 | HOLE | 63 | HOLE | 64 | SHIFTKEYS+ LEFTSHIFT | 65 | 'Z' | 66 | 'X' | 67 | 'C' |
| 68 | 'V' | 69 | 'B' | 6A | 'N' | 6B | 'M' | 6C | '<' | 6D | '>' | 6E | '?' | 6F | SHIFTKEYS+ RIGHTSHIFT |
| 70 | NOP | 71 | 0x7F | 72 | '0' | 73 | NOP | 74 | '.' | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | SHIFTKEYS+ RIGHTSHIFT | 7B | ' ' | 7C | SHIFTKEYS+ LEFTCTRL | 7D | HOLE | 7E | HOLE | 7F | IDLE |

### Micro Switch 103SD32-2 Keyboard

#### Caps Locked

| *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | LF(2) | 03 | LF(3) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF (3) |
| 08 | TF(4) | 09 | TF (5) | 0A | TF(6) | 0B | TF(7) | 0C | TF(8) | 0D | TF(9) | 0E | TF(10) | 0F | TF(11) |
| 10 | TF(12) | 11 | TF(13) | 12 | TF(14) | 13 | c('[') | 14 | HOLE | 15 | RF (1) | 16 | '+' | 17 | '–' |
| 18 | HOLE | 19 | LF(4) | 1A | '\f' | 1B | LF(6) | 1C | HOLE | 1D | SHIFTKEYS+ CAPSLOCK | 1E | '1' | 1F | '2' |
| 20 | '3' | 21 | '4' | 22 | '5' | 23 | '6' | 24 | '7' | 25 | '8' | 26 | '9' | 27 | '0' |
| 28 | '–' | 29 | '=' | 2A | '`' | 2B | '\b' | 2C | HOLE | 2D | '7' | 2E | '8' | 2F | '9' |
| 30 | HOLE | 31 | LF(7) | 32 | STRING+ UPARROW | 33 | LF (9) | 34 | HOLE | 35 | '\t' | 36 | 'Q' | 37 | 'W' |
| 38 | 'E' | 39 | 'R' | 3A | 'T' | 3B | 'Y' | 3C | 'U' | 3D | 'I' | 3E | 'O' | 3F | 'P' |
| 40 | '[' | 41 | ']' | 42 | '_' | 43 | HOLE | 44 | '4' | 45 | '5' | 46 | '6' | 47 | HOLE |
| 48 | STRING+ LEFTARROW | 49 | STRING+ HOMEARROW | 4A | STRING+ RIGHTARROW | 4B | HOLE | 4C | SHIFTKEYS+ SHIFTLOCK | 4D | 'A' | 4E | 'S' | 4F | 'D' |
| 50 | 'F' | 51 | 'G' | 52 | 'H' | 53 | 'J' | 54 | 'K' | 55 | 'L' | 56 | ';' | 57 | ':' |
| 58 | 'I' | 59 | '\r' | 5A | HOLE | 5B | '1' | 5C | '2' | 5D | '3' | 5E | HOLE | 5F | NOSCROLL |
| 60 | STRING+ DOWNARROW | 61 | LF (15) | 62 | HOLE | 63 | HOLE | 64 | SHIFTKEYS+ LEFTSHIFT | 65 | 'Z' | 66 | 'X' | 67 | 'C' |
| 68 | 'V' | 69 | 'B' | 6A | 'N' | 6B | 'M' | 6C | ',' | 6D | '.' | 6E | '/' | 6F | SHIFTKEYS+ RIGHTSHIFT |
| 70 | NOP | 71 | 0x7F | 72 | '0' | 73 | NOP | 74 | '.' | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | SHIFTKEYS+ LEFTCTRL | 7B | ' ' | 7C | SHIFTKEYS RIGHTCTRL | 7D | HOLE | 7E | HOLE | 7F | IDLE |

#### Controlled

| *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | LF(2) | 03 | LF(3) | 04 | HOLE | 05 | TF(1) | 06 | TF(2) | 07 | TF (3) |
| 08 | TF (4) | 09 | TF(5) | 0A | TF(6) | 0B | TF(7) | 0C | TF(8) | 0D | TF(9) | 0E | TF(10) | 0F | TF(11) |
| 10 | TF(02) | 11 | TF(03) | 12 | TF(04) | 13 | c('[') | 14 | HOLE | 15 | RF (0) | 16 | OOPS | 17 | OOPS |
| 18 | HOLE | 19 | LF(4) | 1A | '\f' | 1B | LF (6) | 1C | HOLE | 1D | SHIFTKEYS+ CAPSLOCK | 1E | OOPS | 1F | OOPS |
| 20 | OOPS | 21 | OOPS | 22 | OOPS | 23 | OOPS | 24 | OOPS | 25 | OOPS | 26 | OOPS | 27 | OOPS |
| 28 | OOPS | 29 | c('^') | 2A | c ('@') | 2B | '\b' | 2C | HOLE | 2D | OOPS | 2E | OOPS | 2F | OOPS |
| 30 | HOLE | 31 | LF(7) | 32 | STRING+ UPARROW | 33 | F(9) | 34 | HOLE | 35 | '\t' | 36 | CTRLQ | 37 | c ('W') |
| 38 | c('E') | 39 | c('R') | 3A | c('T') | 3B | c('Y') | 3C | c('U') | 3D | c('I') | 3E | c('O') | 3F | c ('P') |
| 40 | c('[') | 41 | c(']') | 42 | c ('_') | 43 | HOLE | 44 | OOPS | 45 | OOPS | 46 | OOPS | 47 | HOLE |
| 48 | STRING+ LEFTARROW | 49 | STRING+ HOMEARROW | 4A | STRING+ RIGHTARROW | 4B | HOLE | 4C | SHIFTKEYS+ SHIFTLOCK | 4D | c('A') | 4E | CTRLS | 4F | c('D') |
| 50 | c('F') | 51 | c('G') | 52 | c('H') | 53 | c('J') | 54 | c('K') | 55 | c ('L') | 56 | OOPS | 57 | OOPS |
| 58 | c ('\') | 59 | '\r' | 5A | HOLE | 5B | OOPS | 5C | OOPS | 5D | OOPS | 5E | HOLE | 5F | NOSCROLL |
| 60 | STRING+ DOWNARROW | 61 | LF(15) | 62 | HOLE | 63 | HOLE | 64 | SHIFTKEYS+ LEFTSHIFT | 65 | c('Z') | 66 | c('X') | 67 | c('C') |
| 68 | c('V') | 69 | c('B') | 6A | c('N') | 6B | c ('M') | 6C | OOPS | 6D | OOPS | 6E | OOPS | 6F | SHIFTKEYS+ RIGHTSHIFT |
| 70 | NOP | 71 | 0x7F | 72 | OOPS | 73 | NOP | 74 | OOPS | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | SHIFTKEYS+ LEFTCTRL | 7B | '\0' | 7C | SHIFTKEYS+ RIGHTCTRL | 7D | HOLE | 7E | HOLE | 7F | IDLE |

## Micro Switch 103SD32-2 Keyboard

### Key Up

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | OOPS | 03 | OOPS | 04 | HOLE | 05 | OOPS | 06 | OOPS | 07 | OOPS |
| 08 | OOPS | 09 | OOPS | 0A | OOPS | 0B | OOPS | 0C | OOPS | 0D | OOPS | 0E | OOPS | 0F | OOPS |
| 10 | OOPS | 11 | OOPS | 12 | OOPS | 13 | NOP | 14 | HOLE | 15 | OOPS | 16 | NOP | 17 | NOP |
| 18 | HOLE | 19 | OOPS | 1A | NOP | 1B | OOPS | 1C | HOLE | 1D | SHIFTKEYS+ CAPSLOCK | 1E | NOP | 1F | NOP |
| 20 | NOP | 21 | NOP | 22 | NOP | 23 | NOP | 24 | NOP | 25 | NOP | 26 | NOP | 27 | NOP |
| 28 | NOP | 29 | NOP | 2A | NOP | 2B | NOP | 2C | HOLE | 2D | NOP | 2E | NOP | 2F | NOP |
| 30 | HOLE | 31 | OOPS | 32 | NOP | 33 | OOPS | 34 | HOLE | 35 | NOP | 36 | NOP | 37 | NOP |
| 38 | NOP | 39 | NOP | 3A | NOP | 3B | NOP | 3C | NOP | 3D | NOP | 3E | NOP | 3F | NOP |
| 40 | NOP | 41 | NOP | 42 | NOP | 43 | HOLE | 44 | NOP | 45 | NOP | 46 | NOP | 47 | HOLE |
| 48 | NOP | 49 | NOP | 4A | NOP | 4B | HOLE | 4C | SHIFTKEYS+ SHIFTLOCK | 4D | NOP | 4E | NOP | 4F | NOP |
| 50 | NOP | 51 | NOP | 52 | NOP | 53 | NOP | 54 | NOP | 55 | NOP | 56 | NOP | 57 | NOP |
| 58 | NOP | 59 | NOP | 5A | HOLE | 5B | NOP | 5C | NOP | 5D | NOP | 5E | HOLE | 5F | NOP |
| 60 | NOP | 61 | OOPS | 62 | HOLE | 63 | HOLE | 64 | SHIFTKEYS+ LEFTSHIFT | 65 | NOP | 66 | NOP | 67 | NOP |
| 68 | NOP | 69 | NOP | 6A | NOP | 6B | NOP | 6C | NOP | 6D | NOP | 6E | NOP | 6F | SHIFTKEYS+ RIGHTSHIFT |
| 70 | NOP | 71 | NOP | 72 | NOP | 73 | NOP | 74 | NOP | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | SHIFTKEYS+ LEFTCTRL | 7B | NOP | 7C | SHIFTKEYS+ RIGHTCTRL | 7D | HOLE | 7E | HOLE | 7F | RESET |

## VT100-Style Keyboard

### Unshifted

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | HOLE | 04 | HOLE | 05 | HOLE | 06 | HOLE | 07 | HOLE |
| 08 | HOLE | 09 | HOLE | 0A | STRING+ UPARROW | 0B | STRING+ DOWNARROW | 0C | STRING+ LEFTARROW | 0D | STRING+ RIGHTARROW | 0E | HOLE | 0F | TF(1) |
| 10 | TF(2) | 11 | TF(3) | 12 | TF(4) | 13 | c ('[') | 14 | '1' | 15 | '2' | 16 | '3' | 17 | '4' |
| 18 | '5' | 19 | '6' | 1A | '7' | 1B | '8' | 1C | '9' | 1D | '0' | 1E | '-' | 1F | '=' |
| 20 | '`' | 21 | c ('H') | 22 | BUCKYBITS+ METABIT | 23 | '7' | 24 | '8' | 25 | '9' | 26 | '-' | 27 | '\t' |
| 28 | 'q' | 29 | 'w' | 2A | 'e' | 2B | 'r' | 2C | 't' | 2D | 'y' | 2E | 'u' | 2F | 'i' |
| 30 | 'o' | 31 | 'p' | 32 | '[' | 33 | ']' | 34 | 0x7F | 35 | '4' | 36 | '5' | 37 | '6' |
| 38 | ',' | 39 | SHIFTKEYS+ LEFTCTRL | 3A | SHIFTKEYS+ CAPSLOCK | 3B | 'a' | 3C | 's' | 3D | 'd' | 3E | 'f' | 3F | 'g' |
| 40 | 'h' | 41 | 'j' | 42 | 'k' | 43 | 'l' | 44 | ';' | 45 | '\'' | 46 | '\r' | 47 | '\' |
| 48 | '1' | 49 | '2' | 4A | '3' | 4B | NOP | 4C | NOSCROLL | 4D | SHIFTKEYS+ LEFTSHIFT | 4E | 'z' | 4F | 'x' |
| 50 | 'c' | 51 | 'v' | 52 | 'b' | 53 | 'n' | 54 | 'm' | 55 | ',' | 56 | '.' | 57 | '/' |
| 58 | SHIFTKEYS+ RIGHTSHIFT | 59 | '\n' | 5A | '0' | 5B | HOLE | 5C | '.' | 5D | '\r' | 5E | HOLE | 5F | HOLE |
| 60 | HOLE | 61 | HOLE | 62 | ' ' | 63 | HOLE | 64 | HOLE | 65 | HOLE | 66 | HOLE | 67 | HOLE |
| 68 | HOLE | 69 | HOLE | 6A | HOLE | 6B | HOLE | 6C | HOLE | 6D | HOLE | 6E | HOLE | 6F | HOLE |
| 70 | HOLE | 71 | HOLE | 72 | HOLE | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | HOLE | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | IDLE |

## VT100-Style Keyboard

### Shifted

| *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | HOLE | 04 | HOLE | 05 | HOLE | 06 | HOLE | 07 | HOLE |
| 08 | HOLE | 09 | HOLE | 0A | STRING+ UPARROW | 0B | STRING+ DOWNARROW | 0C | STRING+ LEFTARROW | 0D | STRING+ RIGHTARROW | 0E | HOLE | 0F | TF(1) |
| 10 | TF(2) | 11 | TF(3) | 12 | TF(4) | 13 | c ('[') | 14 | '!' | 15 | '@' | 16 | '#' | 17 | '$' |
| 18 | '%' | 19 | '^' | 1A | '&' | 1B | '*' | 1C | '(' | 1D | ')' | 1E | '_' | 1F | '+' |
| 20 | '~' | 21 | c ('H') | 22 | BUCKYBITS+ METABIT | 23 | '7' | 24 | '8' | 25 | '9' | 26 | '–' | 27 | '\t' |
| 28 | 'Q' | 29 | 'W' | 2A | 'E' | 2B | 'R' | 2C | 'T' | 2D | 'Y' | 2E | 'U' | 2F | 'I' |
| 30 | 'O' | 31 | 'P' | 32 | '{' | 33 | '}' | 34 | 0x7F | 35 | '4' | 36 | '5' | 37 | '6' |
| 38 | ',' | 39 | SHIFTKEYS+ LEFTCTRL | 3A | SHIFTKEYS+ CAPSLOCK | 3B | 'A' | 3C | 'S' | 3D | 'D' | 3E | 'F' | 3F | 'G' |
| 40 | 'H' | 41 | 'J' | 42 | 'K' | 43 | 'L' | 44 | ':' | 45 | '"' | 46 | '\r' | 47 | '|' |
| 48 | '1' | 49 | '2' | 4A | '3' | 4B | NOP | 4C | NOSCROLL | 4D | SHIFTKEYS+ LEFTSHIFT | 4E | 'Z' | 4F | 'X' |
| 50 | 'C' | 51 | 'V' | 52 | 'B' | 53 | 'N' | 54 | 'M' | 55 | '<' | 56 | '>' | 57 | '?' |
| 58 | SHIFTKEYS+ RIGHTSHIFT | 59 | '\n' | 5A | '0' | 5B | HOLE | 5C | '.' | 5D | '\r' | 5E | HOLE | 5F | HOLE |
| 60 | HOLE | 61 | HOLE | 62 | ' ' | 63 | HOLE | 64 | HOLE | 65 | HOLE | 66 | HOLE | 67 | HOLE |
| 68 | HOLE | 69 | HOLE | 6A | HOLE | 6B | HOLE | 6C | HOLE | 6D | HOLE | 6E | HOLE | 6F | HOLE |
| 70 | HOLE | 71 | HOLE | 72 | HOLE | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | HOLE | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | IDLE |

### Caps Locked

| *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* | *Key* | *Value* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | HOLE | 04 | HOLE | 05 | HOLE | 06 | HOLE | 07 | HOLE |
| 08 | HOLE | 09 | HOLE | 0A | STRING+ UPARROW | 0B | STRING+ DOWNARROW | 0C | STRING+ LEFTARROW | 0D | STRING+ RIGHTARROW | 0E | HOLE | 0F | TF(1) |
| 10 | TF(2) | 11 | TF(3) | 12 | TF(4) | 13 | c ('[') | 14 | '1' | 15 | '2' | 16 | '3' | 17 | '4' |
| 18 | '5' | 19 | '6' | 1A | '7' | 1B | '8' | 1C | '9' | 1D | '0' | 1E | '–' | 1F | '=' |
| 20 | '^' | 21 | c ('H') | 22 | BUCKYBITS+ METABIT | 23 | '7' | 24 | '8' | 25 | '9' | 26 | '–' | 27 | '\t' |
| 28 | 'Q' | 29 | 'W' | 2A | 'E' | 2B | 'R' | 2C | 'T' | 2D | 'Y' | 2E | 'U' | 2F | 'I' |
| 30 | 'O' | 31 | 'P' | 32 | '[' | 33 | ']' | 34 | 0x7F | 35 | '4' | 36 | '5' | 37 | '6' |
| 38 | ',' | 39 | SHIFTKEYS+ LEFTCTRL | 3A | SHIFTKEYS+ CAPSLOCK | 3B | 'A' | 3C | 'S' | 3D | 'D' | 3E | 'F' | 3F | 'G' |
| 40 | 'H' | 41 | 'J' | 42 | 'K' | 43 | 'L' | 44 | ';' | 45 | '\'' | 46 | '\r' | 47 | '\' |
| 48 | '1' | 49 | '2' | 4A | '3' | 4B | NOP | 4C | NOSCROLL | 4D | SHIFTKEYS+ LEFTSHIFT | 4E | 'Z' | 4F | 'X' |
| 50 | 'C' | 51 | 'V' | 52 | 'B' | 53 | 'N' | 54 | 'M' | 55 | ',' | 56 | '.' | 57 | '/' |
| 58 | SHIFTKEYS+ RIGHTSHIFT | 59 | '\n' | 5A | '0' | 5B | HOLE | 5C | '.' | 5D | '\r' | 5E | HOLE | 5F | HOLE |
| 60 | HOLE | 61 | HOLE | 62 | ' ' | 63 | HOLE | 64 | HOLE | 65 | HOLE | 66 | HOLE | 67 | HOLE |
| 68 | HOLE | 69 | HOLE | 6A | HOLE | 6B | HOLE | 6C | HOLE | 6D | HOLE | 6E | HOLE | 6F | HOLE |
| 70 | HOLE | 71 | HOLE | 72 | HOLE | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | HOLE | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | IDLE |

### VT100-Style Keyboard

#### Controlled

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | HOLE | 04 | HOLE | 05 | HOLE | 06 | HOLE | 07 | HOLE |
| 08 | HOLE | 09 | HOLE | 0A | STRING+ UPARROW | 0B | STRING+ DOWNARROW | 0C | STRING+ LEFTARROW | 0D | STRING+ RIGHTARROW | 0E | HOLE | 0F | TF(1) |
| 10 | TF(2) | 11 | TF(3) | 12 | TF(4) | 13 | c('[') | 14 | '1' | 15 | c ('@') | 16 | '3' | 17 | '4' |
| 18 | '5' | 19 | c('^') | 1A | '7' | 1B | '8' | 1C | '9' | 1D | '0' | 1E | c ('_') | 1F | '=' |
| 20 | c('~') | 21 | c ('H') | 22 | BUCKYBITS+ METABIT | 23 | '7' | 24 | '8' | 25 | '9' | 26 | '-' | 27 | '\t' |
| 28 | CTRLQ | 29 | c('W') | 2A | c('E') | 2B | c('R') | 2C | c('T') | 2D | c('Y') | 2E | c('U') | 2F | c ('I') |
| 30 | c('O') | 31 | c('P') | 32 | c('[') | 33 | c (']') | 34 | 0x7F | 35 | '4' | 36 | '5' | 37 | '6' |
| 38 | ',' | 39 | SHIFTKEYS+ LEFTCTRL | 3A | SHIFTKEYS+ CAPSLOCK | 3B | c('A') | 3C | CTRLS | 3D | c('D') | 3E | c('F') | 3F | c ('G') |
| 40 | c('H') | 41 | c('J') | 42 | c('K') | 43 | c('L') | 44 | ':' | 45 | '"' | 46 | '\r' | 47 | c ('\') |
| 48 | '1' | 49 | '2' | 4A | '3' | 4B | NOP | 4C | NOSCROLL | 4D | SHIFTKEYS+ LEFTSHIFT | 4E | c('Z') | 4F | c ('X') |
| 50 | c('C') | 51 | c('V') | 52 | c('B') | 53 | c('N') | 54 | c('M') | 55 | ',' | 56 | '.' | 57 | c ('_') |
| 58 | SHIFTKEYS+ RIGHTSHIFT | 59 | '\n' | 5A | '0' | 5B | HOLE | 5C | '.' | 5D | HOLE | 5E | HOLE | 5F | HOLE |
| 60 | HOLE | 61 | HOLE | 62 | c (' ') | 63 | HOLE | 64 | HOLE | 65 | HOLE | 66 | HOLE | 67 | HOLE |
| 68 | HOLE | 69 | HOLE | 6A | HOLE | 6B | HOLE | 6C | HOLE | 6D | HOLE | 6E | HOLE | 6F | HOLE |
| 70 | HOLE | 71 | HOLE | 72 | HOLE | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | HOLE | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | IDLE |

#### Key Up

| Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value | Key | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | HOLE | 01 | BUCKYBITS+ SYSTEMBIT | 02 | HOLE | 03 | HOLE | 04 | HOLE | 05 | HOLE | 06 | HOLE | 07 | HOLE |
| 08 | HOLE | 09 | HOLE | 0A | NOP | 0B | NOP | 0C | NOP | 0D | NOP | 0E | HOLE | 0F | OOPS |
| 10 | OOPS | 11 | OOPS | 12 | OOPS | 13 | NOP | 14 | NOP | 15 | NOP | 16 | NOP | 17 | NOP |
| 18 | NOP | 19 | NOP | 1A | NOP | 1B | NOP | 1C | NOP | 1D | NOP | 1E | NOP | 1F | NOP |
| 20 | NOP | 21 | NOP | 22 | BUCKYBITS+ METABIT | 23 | NOP | 24 | NOP | 25 | NOP | 26 | NOP | 27 | NOP |
| 28 | NOP | 29 | NOP | 2A | NOP | 2B | NOP | 2C | NOP | 2D | NOP | 2E | NOP | 2F | NOP |
| 30 | NOP | 31 | NOP | 32 | NOP | 33 | NOP | 34 | NOP | 35 | NOP | 36 | NOP | 37 | NOP |
| 38 | NOP | 39 | SHIFTKEYS+ LEFTCTRL | 3A | SHIFTKEYS+ CAPSLOCK | 3B | NOP | 3C | NOP | 3D | NOP | 3E | NOP | 3F | NOP |
| 40 | NOP | 41 | NOP | 42 | NOP | 43 | NOP | 44 | NOP | 45 | NOP | 46 | NOP | 47 | NOP |
| 48 | NOP | 49 | NOP | 4A | NOP | 4B | NOP | 4C | NOP | 4D | SHIFTKEYS+ LEFTSHIFT | 4E | NOP | 4F | NOP |
| 50 | NOP | 51 | NOP | 52 | NOP | 53 | NOP | 54 | NOP | 55 | NOP | 56 | NOP | 57 | NOP |
| 58 | SHIFTKEYS+ RIGHTSHIFT | 59 | NOP | 5A | NOP | 5B | HOLE | 5C | NOP | 5D | NOP | 5E | HOLE | 5F | HOLE |
| 60 | HOLE | 61 | HOLE | 62 | NOP | 63 | HOLE | 64 | HOLE | 65 | HOLE | 66 | HOLE | 67 | HOLE |
| 68 | HOLE | 69 | HOLE | 6A | HOLE | 6B | HOLE | 6C | HOLE | 6D | HOLE | 6E | HOLE | 6F | HOLE |
| 70 | HOLE | 71 | HOLE | 72 | HOLE | 73 | HOLE | 74 | HOLE | 75 | HOLE | 76 | HOLE | 77 | HOLE |
| 78 | HOLE | 79 | HOLE | 7A | HOLE | 7B | HOLE | 7C | HOLE | 7D | HOLE | 7E | HOLE | 7F | RESET |

## SEE ALSO

**click**(1), **oldsetkeys**(1), **kbd**(4S), **termio**(4), **win**(4S)
*The SunView System Programmer's Guide— Appendix: Writing a Virtual User Input Device Driver* (describes **firm_event** format)

**NAME**

kbd – Sun keyboard

**CONFIG**

None; included in standard system.

**DESCRIPTION**

The **kbd** device provides access to the Sun Workstation keyboard. When opened, it provides access to the standard keyboard device for the workstation (attached either to a CPU serial or parallel port). It is a multiplexing driver; a stream referring to the standard keyboard device, with the **kb**(4M) and **ttcompat**(4M) STREAMS modules pushed on top of that device, is linked below it. Normally, this device passes input to the "workstation console" driver, which is linked above a special minor device of **kbd**, so that keystrokes appear as input on **/dev/console**; the **KIOCSDIRECT ioctl** must be used to direct input towards or away from the **/dev/kbd** device.

**IOCTLS**

**KIOCSDIRECT** The argument is a pointer to an **int**. If the value in the **int** pointed to by the argument is 1, subsequent keystrokes typed on the system keyboard will sent to **/dev/kbd**; if it is 0, subsequent keystrokes will be sent to the "workstation console" device. When the last process that has **/dev/kbd** open closes it, if keystrokes had been sent to **/dev/kbd** they are redirected back to the "workstation console" device.

**KIOCGDIRECT** The argument is a pointer to an **int**. If keystrokes are currently being sent to **/dev/kbd**, 1 is stored in the **int** pointed to by the argument; if keystrokes are currently being sent to the "workstation console" device, 0 is stored there.

**FILES**

**/dev/kbd**

**SEE ALSO**

**console**(4S), **kb**(4M), **ttcompat**(4M), **win**(4S), **zs**(4S)

**NAME**

ldterm – standard terminal STREAMS module

**CONFIG**

None; included by default.

**SYNOPSIS**

```
#include <sys/stream.h>
#include <sys/stropt.h>

ioctl(fd, I_PUSH, "ldterm");
```

**DESCRIPTION**

**ldterm** is a STREAMS module that provides most of the **termio (4)** terminal interface. This module does not perform the low-level device control functions specified by flags in the **c_cflag** word of the **termios** structure or by the **IGNBRK, IGNPAR, PARMRK,** or **INPCK** flags in the **c_iflag** word of the **termios** structure; those functions must be performed by the driver or by modules pushed below the **ldterm** module. All other **termio** functions are performed by **ldterm**; some of them, however, require the cooperation of the driver or modules pushed below **ldterm**, and may not be performed in some cases. These include the **IXOFF** flag in the **c_iflag** word and the delays specified in the **c_oflag** word.

**Read-side Behavior**

Various types of STREAMS messages are processed as follows:

**M_BREAK** When this message is received, either an interrupt signal is generated, or the message is treated as if it were an **M_DATA** message containing a single ASCII NUL character, depending on the state of the **BRKINT** flag.

**M_DATA** These messages are normally processed using the standard **termio** input processing. If the **ICANON** flag is set, a single input record (''line'') is accumulated in an internal buffer, and sent upstream when a line-terminating character is received. If the **ICANON** flag is not set, other input processing is performed and the processed data is passed upstream.

If output is to be stopped or started as a result of the arrival of characters, **M_STOP** and **M_START** messages are sent downstream, respectively. If the **IXOFF** flag is set, and input is to be stopped or started as a result of flow-control considerations, **M_STOPI** and **M_STARTI** messages are sent downstream, respectively.

**M_DATA** messages are sent downstream, as necessary, to perform echoing.

If a signal is to be generated, a **M_FLUSH** message with a flag byte of **FLUSHR** is placed on the read queue, and if the signal is also to flush output a **M_FLUSH** message with a flag byte of **FLUSHW** is sent downstream.

**M_CTL** If the first byte of the message is **MC_NOCANON**, the input processing normally performed on **M_DATA** messages is disabled, and those messages are passed upstream unmodified; this is for the use of modules or drivers that perform their own input processing, such as a pseudo-terminal in **TIOCREMOTE** mode connected to a program that performs this processing. If the first byte of the message is **MC_DOCANON**, the input processing is enabled. Otherwise, the message is ignored; in any case, the message is passed upstream.

**M_FLUSH** The read queue of the module is flushed of all its data messages, and all data in the record being accumulated is also flushed. The message is passed upstream.

**M_HANGUP** Data is flushed as it is for a **M_FLUSH** message, and **M_FLUSH** messages with a flag byte of **FLUSHRW** are sent upstream and downstream. Then an **M_PCSIG** message is sent upstream with a signal of **SIGCONT**, followed by the **M_HANGUP** message.

**M_IOCACK** The data contained within the message, which is to be returned to the process, is augmented if necessary, and the message is passed upstream.

All other messages are passed upstream unchanged.

**Write-side behavior**

Various types of STREAMS messages are processed as follows:

**M_FLUSH** The write queue of the module is flushed of all its data messages, and the message is passed downstream.

**M_IOCTL** The function to be performed for this **ioctl** by the **ldterm** module is performed, and the message is passed downstream in most cases. The **TCFLSH** and **TCXONC** **ioctls** can be performed entirely in this module, so the reply is sent upstream and the message is not passed downstream.

**M_DATA** If the **OPOST** flag is set, or both the **XCASE** and **ICANON** flags are set, output processing is performed and the processed message is passed downstream, along with any **M_DELAY** messages generated. Otherwise, the message is passed downstream without change.

All other messages are passed downstream unchanged.

## IOCTLS

The following **ioctls** are processed by the **ldterm** module. All others are passed downstream.

**TCGETS**
**TCGETA** The message is passed downstream; if an acknowledgment is seen, the data provided by the driver and modules downstream is augmented and the acknowledgement is passed upstream.

**TCSETS**
**TCSETSW**
**TCSETSF**
**TCSETA**
**TCSETAW**
**TCSETAF** The parameters that control the behavior of the **ldterm** module are changed. If a mode change requires options at the stream head to be changed, a **M_SETOPT** message is sent upstream. If the **ICANON** flag is turned on or off, the read mode at the stream head is changed to message-nondiscard or byte-stream mode, respectively. If it is turned on, the **vmin** and **vtime** values at the stream head are set to 1 and 0, respectively; if it is turned on, they are set to the values specified by the **ioctl**. The **vmin** and **vtime** values are also set if **ICANON** is off and the values are changed by the **ioctl**. If the **TOSTOP** flag is turned on or off, the **tostop** mode at the stream head is turned on or off, respectively.

**TCFLSH** If the argument is 0, an **M_FLUSH** message with a flag byte of **FLUSHR** is sent downstream and placed on the read queue. If the argument is 1, the write queue is flushed of all its data messages and a **M_FLUSH** message with a flag byte of **FLUSHW** is sent upstream and downstream. If the argument is 2, the write queue is flushed of all its data messages and a **M_FLUSH** message with a flag byte of **FLUSHRW** is sent downstream and placed on the read queue.

**TCXONC** If the argument is 0, and output is not already stopped, an **M_STOP** message is sent downstream. If the argument is 1, and output is stopped, an **M_START** message is sent downstream. If the argument is 2, and input is not already stopped, an **M_STOPI** message is sent downstream. If the argument is 3, and input is stopped, an **M_STARTI** message is sent downstream.

## SEE ALSO

**console**(4S), **mcp**(4S), **mti**(4S), **pty**(4), **termio**(4), **ttcompat**(4M), **zs**(4S)

## NAME

le – Sun-3/50, Sun-3/60 10MB Ethernet interface

## CONFIG

**device le0 at obio ? csr**

## DESCRIPTION

The **le** interface provides access to a 10 Mb/s Ethernet network through a Sun-3 controller using the AMD LANCE (Local Area Network Controller for Ethernet) Am7990 chip. For a general description of network interfaces see **if**(4N).

The synopsis line above specifies the first and only Ethernet controller on a Sun-3/50 system.

## SEE ALSO

**if**(4N), **kb**(4S), **tty_compact**(4)

## DIAGNOSTICS

**le%d: transmitter frozen — resetting**
A bug in the LANCE chip has stopped the chip's transmitter section. The driver has detected this condition and reinitialized the chip.

**le%d: out of mbufs: output packet dropped**
The driver has run out of memory to use to buffer packets on output. The packet being transmitted at the time of occurrence is lost. This error is usually symptomatic of trouble elsewhere in the kernel.

**le%d: stray transmitter interrupt**
The LANCE chip has signalled that it completed transmitting a packet but the driver has sent no such packet.

**le%d: LANCE Rev C/D Extra Byte(s) bug; Packet dropped**
The LANCE chip's internal silo pointers have become misaligned. This error arises from a chip bug.

**le%d: trailer error**
An incoming packet claimed to have a trailing header but did not.

**le%d: runt packet**
An incoming packet's size was below the Ethernet minimum transmission size.

**le%d: Receive buffer error - BUFF bit set in rmd**
This error "should never happen," as it occurs only in conjunction with a LANCE feature that the driver does not use.

**le%d: Received packet with STP bit in rmd cleared**
The driver has received a packet that straddles multiple receive buffers and therefore consumes more than one of the LANCE chip's receive descriptors. Provided that all stations on the Ethernet are operating according to the Ethernet specification, this error "should never happen," since the driver allocates its receive buffers to be large enough to hold packets of the largest permitted size. Most likely, some other station on the net is transmitting packets whose lengths exceed the maximum permitted for Ethernet.

**le%d: Received packet with ENP bit in rmd cleared**
The driver has received a packet that straddles multiple receive buffers and therefore consumes more than one of the LANCE chip's receive descriptors. Provided that all stations on the Ethernet are operating according to the Ethernet specification, this error "should never happen," since the driver allocates its receive buffers to be large enough to hold packets of the largest permitted size. The most likely cause of the message is that some other station on the net is transmitting packets whose lengths exceed the maximum permitted for Ethernet.

**le%d: Transmit buffer error - BUFF bit set in tmd**
Excessive bus contention has prevented the LANCE chip from gathering packet contents quickly enough to sustain the packet's transmission over the Ethernet. The affected packet is lost.

**le%d: Transmit late collision - Net problem?**
A packet collision has occurred after the channel's slot time has elapsed. This error usually indicates faulty hardware elsewhere on the net.

**le%d: No carrier - transceiver cable problem?**
The LANCE chip has lost input to its carrier detect pin while trying to transmit a packet.

**le%d: Transmit retried more than 16 times - net jammed**
Network activity has become so intense that sixteen successive transmission attempts failed, the LANCE chip gave up on the current packet.

**le%d: missed packet**
The driver has dropped an incoming packet because it had no buffer space for it.

**le%d: Babble error - sent a packet longer than the maximum length**
While transmitting a packet, the LANCE chip has noticed that the packet's length exceeds the maximum allowed for Ethernet. This error indicates a kernel bug.

**le%d: Memory Error! Ethernet chip memory access timed out**
The LANCE chip timed out while trying to acquire the bus for a DVMA transfer.

**le%d: Reception stopped**
Because of some other error, the receive section of the LANCE chip shut down and had to be restarted.

**le%d: Transmission stopped**
Because of some other error, the transmit section of the LANCE chip shut down and had to be restarted.

## NAME

lo – software loopback network interface

## SYNOPSIS

**pseudo-device loop**

## DESCRIPTION

The **loop** device is a software loopback network interface; see **if**(4N) for a general description of network interfaces.

The **loop** interface is used for performance analysis and software testing, and to provide guaranteed access to Internet protocols on machines with no local network interfaces. A typical application is the **comsat**(8C) server which accepts notification of mail delivery through a particular port on the loopback interface.

By default, the loopback interface is accessible at Internet address 127.0.0.1 (non-standard); this address may be changed with the **SIOCSIFADDR** ioctl.

## SEE ALSO

**if**(4N), **inet**(4F), **comsat**(8C)

## DIAGNOSTICS

**lo%d: can't handle af%d**

The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

## BUGS

It should handle all address and protocol families. An approved network address should be reserved for this interface.

## NAME

lofs – loopback virtual file system

## CONFIG

**options LOFS**

## SYNOPSIS

**#include <sys/mount.h>**
**mount(MOUNT_LOFS, virtual, flags, dir);**

*virtual* is the mount point for the virtual file system. *dir* is the pathname of the existing file system. *flags* is either 0 or **M_RDONLY**. The **M_RDONLY** flag forces all accesses in the new name space to be read-only; without it, accesses are the same as for the underlying file system. All other **mount**(2) flags are preserved from the underlying file systems.

## DESCRIPTION

The loopback filesystem device allows new, virtual, file systems to be created, which provide access to existing files using alternate pathnames. Once the virtual file system is created, other file systems can be mounted within it without affecting the original file system. File systems that are subsequently mounted onto the original filesystem, however, *are* visible to the virtual file system, unless or until the corresponding mount point in the virtual file system is covered by a file system mounted there.

For instance, a loopback mount of / onto **/tmp/newroot** allows the entire filesystem hierarchy to appear as if it were duplicated under **/tmp/newroot**, including any file systems mounted from remote NFS servers. All files would then be accessible either from a pathname relative to /, or from a pathname relative to **/tmp/newroot** until such time as a file system is mounted in **/tmp/newroot**, or any of its subdirectories.

Loopback mounts of / can be performed in conjunction with the **chroot**(2) system call, to provide a complete virtual filesystem to a process or family of processes.

Recursive traversal of loopback mount points is not allowed; after the loopback mount of **/tmp/newroot**, the file **/tmp/newroot/tmp/newroot** does not contain yet another filesystem hierarchy; rather, it appears just as **/tmp/newroot** did before the loopback mount was performed (say, as an empty directory).

## SEE ALSO

**chroot**(2), **mount**(2)

## BUGS

Because only directories can be mounted or mounted on, the structure of a virtual file system can only be modified at directories.

Loopback mounts must be used with care; the potential for confusing users and applications is enormous.

## NAME

mcp, alm – Sun MCP Multiprotocol Communications Processor/ALM-2 Asynchronous Line Multiplexer

## CONFIG — SUN-3 SYSTEM

### MCP

**device mcp0 at vme32d32 ? csr 0x1000000 flags 0x1ffff priority 4 vector mcpintr 0x8b**
**device mcp1 at vme32d32 ? csr 0x1010000 flags 0x1ffff priority 4 vector mcpintr 0x8a**
**device mcp2 at vme32d32 ? csr 0x1020000 flags 0x1ffff priority 4 vector mcpintr 0x89**
**device mcp3 at vme32d32 ? csr 0x1030000 flags 0x1ffff priority 4 vector mcpintr 0x88**

### ALM-2

**pseudo-device mcpa64**

## SYNOPSIS

**#include <fcntl.h>**
**#include <sys/termios.h>**
**open("/dev/tty*xy*", mode);**
**open("/dev/ttyd*n*", mode);**
**open("/dev/cua*n*", mode);**

## DESCRIPTION (MCP)

The Sun MCP (Multiprotocol Communications Processor) supports up to four synchronous serial lines in conjunction with SunLink™ Multiple Communication Protocol products.

## DESCRIPTION (ALM-2)

The Sun ALM-2 Asynchronous Line Multiplexer provides 16 asynchronous serial communication lines with modem control and one Centronics-compatible parallel printer port.

Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the IGNBRK, IGNPAR, PARMRK, or INPCK flags in the **c_iflag** word of the **termios** structure are performed by the **mcp** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mcp0** would treat line **/dev/ttyh2** in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* represents the physical board as one of the characters **h**, **i**, **j**, or **k**, and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is **/dev/ttyh0**, and the sixteenth line on the third board is **/dev/ttyjf**.)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed **/dev/ttyd***n*, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua***n*, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line cannot be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding **/dev/cua***n* line cannot be opened. This allows a modem to be

attached to e.g. **/dev/ttyd0** (renamed from **/dev/ttyh0**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

IOCTLS

The standard set of **termio ioctl()** calls are supported by the ALM-2.

If the CRTSCTS flag in the **c_cflag** is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the CRTSCTS flag is clear, the state of CTS has no effect. Breaks can be generated by the TCSBRK, TIOCSBRK, and TIOCCBRK **ioctl()** calls. The modem control lines TIOCM_CAR, TIOCM_CTS, TIOCM_RTS, and TIOCM_DTR are provided.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

ERRORS

An **open()** on a **/dev/tty*** or a **/dev/cu*** device will fail if:

| | |
|---|---|
| ENXIO | The unit being opened does not exist. |
| EBUSY | The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open. |
| EBUSY | The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl()** call. |
| EINTR | The open was interrupted by the delivery of a signal. |

DESCRIPTION (PRINTER PORT)

The printer port is Centronics-compatible and is suitable for most common parallel printers. Devices attached to this interface are normally handled by the line printer spooling system, and should not be accessed directly by the user.

Minor device numbers in the range 64 – 67 access the printer port, and the recommended naming is **/dev/mcpp[0-3]**.

IOCTLS

Various control flags and status bits may be fetched and set on an MCP printer port. The following flags and status bits are supported; they are defined in **<sundev/mcpcmd.h>**:

| | | |
|---|---|---|
| MCPRIGNSLCT | 0x02 | set if interface ignoring SLCT– on open |
| MCPRDIAG | 0x04 | set if printer is in self-test mode |
| MCPRVMEINT | 0x08 | set if VME bus interrupts enabled |
| MCPRINTPE | 0x10 | print message when out of paper |
| MCPRINTSLCT | 0x20 | print message when printer offline |
| MCPRPE | 0x40 | set if device ready, cleared if device out of paper |
| MCPRSLCT | 0x80 | set if device online (Centronics SLCT asserted) |

The flags MCPRINTSLCT, MCPRINTPE, and MCPRDIAG may be changed; the other bits are status bits and may not be changed.

The **ioctl()** calls supported by MCP printer ports are listed below.

| | |
|---|---|
| MCPIOGPR | The argument is a pointer to an **unsigned char**. The printer flags and status bits are stored in the **unsigned char** pointed to by the argument. |
| MCPIOSPR | The argument is a pointer to an **unsigned char**. The printer flags are set from the **unsigned char** pointed to by the argument. |

ERRORS

Normally, the interface only reports the status of the device when attempting an **open**(2V) call. An **open()** on a **/dev/mcpp*** device will fail if:

ENXIO The unit being opened does not exist.

EIO The device is offline or out of paper.

Bit 17 of the configuration **flags** may be specified to say that the interface should ignore Centronics SLCT– and RDY/PE– when attempting to open the device, but this is normally useful only for configuration and troubleshooting: if the SLCT– and RDY lines are not asserted during an actual data transfer (as with a **write**(2V) call), no data is transferred.

FILES

| | |
|---|---|
| **/dev/mcpp[0-3]** | parallel printer port |
| **/dev/tty[h-k][0-9a-f]** | hardwired tty lines |
| **/dev/ttyd[0-9a-f]** | dialin tty lines |
| **/dev/cua[0-9a-f]** | dialout tty lines |

SEE ALSO

**tip**(1C), **uucp**(1C), **mti**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S)

DIAGNOSTICS

Most of these diagnostics "should never happen;" their occurrence usually indicates problems elsewhere in the system as well.

**mcpa*n*: silo overflow.**
More than *n* characters (*n* very large) have been received by the **mcp** hardware without being read by the software.

*****port *n* supports RS449 interface*****
Probably an incorrect jumper configuration. Consult the hardware manual.

**mcp port *n* receive buffer error**
The **mcp** encountered an error concerning the synchronous receive buffer.

**Printer on mcpp*n* is out of paper**
**Printer on mcpp*n* paper ok**
**Printer on mcpp*n* is offline**
**Printer on mcpp*n* online**
Assorted printer diagnostics, if enabled as discussed above.

## NAME

mem, kmem, vme16d16, vme24d16, vme32d16, vme16d32, vme24d32, vme32d32, mbmem, mbio, atbus, zero, eeprom – main memory and bus I/O space

## CONFIG

None; included with standard system.

## DESCRIPTION

These devices are special files that map memory and bus I/O space. They may be read, written, seeked and (except for **kmem**) memory-mapped. See **read**(2V), **write**(2V), **mmap**(2), and **directory**(3),

**mem** is a special file that is an image of the physical memory of the computer. It may be used, for example, to examine (and even to patch) the system.

**kmem** is a special file that is an image of the kernel virtual memory of the system.

**kmem** is a special file which is a source of private zero pages.

### Sun-2 and Sun-3 System

**vme16d16** (also known as **vme16**) is a special file that is an image of VMEbus 16-bit addresses with 16-bit data. **vme16** address space extends from 0 to 64K.

**vme24d16** (also known as **vme24**) is a special file that is an image of VMEbus 24-bit addresses with 16-bit data. **vme24** address space extends from 0 to 16 Megabytes. The VME 16-bit address space overlaps the top 64K of the 24-bit address space.

### Sun-3 VMEbus

**vme32d16** is a special file that is an image of VMEbus 32-bit addresses with 16-bit data.

**vme16d32** is a special file that is an image of VMEbus 16-bit addresses with 32-bit data.

**vme24d32** is a special file that is an image of VMEbus 24-bit addresses with 32-bit data.

**vme32d32** (also known as **vme32**) is a special file that is an image of VMEbus 32-bit addresses with 32-bit data. **vme32** address space extends from 0 to 4 Giggabytes. The VME 24-bit address space overlaps the top 16 Megabytes of the 32-bit address space.

**vme*** type special files can only be accessed in VME based systems.

### Sun-2 MultibuS

**mbmem** is a special file that is an image of the Multibus memory of the system. Multibus memory is in the range from 0 to 16 Megabytes. **mbmem** can only be accessed in Multibus based systems.

**mbio** is a special file that is an image of the Multibus I/O space. Multibus I/O space extends from 0 to 64K. **mbio** can only be accessed in Multibus based systems.

When reading and writing **mbmem** and **mbio** odd counts or offsets cause byte accesses and even counts and offsets cause word accesses.

### Sun386i

**atbus** is a special file that is an image of the AT bus space. It extends from 0 to 16 Megabytes.

**eeprom** is a special file that is an image of the NVRAM. It extends from 0 to 2Kb.

## FILES

**/dev/mem**
**/dev/kmem**
**/dev/mbmem**
**/dev/mbio**
**/dev/vme16d16**
**/dev/vme16**
**/dev/vme24d16**
**/dev/vme24**
**/dev/vme32d16**

**/dev/vme16d32**
**/dev/vme24d32**
**/dev/vme32d32**
**/dev/vme32**
**/dev/atbus**
**/dev/zero**
**/dev/eeprom**

**SEE ALSO**

**mmap**(2), **read**(2V), **write**(2V), **directory**(3)

**NAME**

mouse – Sun mouse

**CONFIG**

None; included in standard system.

**DESCRIPTION**

The **mouse** indirect device provides access to the Sun Workstation mouse. When opened, it redirects operations to the standard mouse device for the workstation (attached either to a CPU serial or parallel port), and pushes the **ms**(4M) and **ttcompat**(4M) STREAMS modules on top of that device.

**FILES**

**/dev/mouse**

**SEE ALSO**

**ms**(4M), **ttcompat**(4M), **win**(4S), **zs**(4S)

## NAME

ms – Sun mouse STREAMS module

## CONFIG

**pseudo-device***ms***n**

## SYNOPSIS

```
#include <sys/stream.h>
#include <sys/stropt.h>
#include <sundev/vuid_event.h>
#include <sundev/msio.h>
ioctl(fd, I_PUSH, "ms");
```

## DESCRIPTION

The **ms** STREAMS module processes byte streams generated by mice attached to a CPU serial or parallel port. When this module is pushed onto a stream, it sends a **TCSETSF** ioctl downstream, setting the baud rate to 1200 baud and the character size to 8 bits, and enabling the receiver. All other flag words are cleared. It assumes only that the **termios**(4) functions provided by the **zs**(4S) driver are supported; no other functions need be supported.

The mouse is expected to generate a stream of bytes encoding mouse motions and changes in the state of the buttons.

Each mouse sample in the byte stream consists of three bytes: the first byte gives the button state with value 0x87|~*but*, where *but* is the low three bits giving the mouse buttons, where a 0 (zero) bit means that a button is pressed, and a 1 (one) bit means a button is not pressed. Thus if the left button is down the value of this sample is 0x83, while if the right button is down the byte is 0x86.

The next two bytes of each sample give the *x* and *y* deltas of this sample as signed bytes. The mouse uses a lower-left coordinate system, so moves to the right on the screen yield positive *x* values and moves down the screen yield negative *y* values.

The beginning of a sample is identifiable because the delta's are constrained to not have values in the range 0x80-0x87.

A stream with **ms** pushed onto it can be used as a device that emits *firm_events* as specified by the protocol of a *Virtual User Input Device*. It understands **VUIDSFORMAT, VUIDGFORMAT, VUIDSADDR** and **VUIDGADDR** ioctls (see reference below).

## IOCTLS

**ms** responds to the following *ioctl*s, as defined in **<sundev/msio.h>** and **<sundev/vuid_event.h>**. All other *ioctl*s are passed downstream. As **ms** sets the parameters of the serial port when it is opened, no **termios**(4) *ioctl*s should be performed on a stream with **ms** on it, as **ms** expects the device parameters to remain as it set them.

The **MSIOGETPARMS** and **MSIOSETPARMS** calls use a structure of type **Ms_parms**, which is a structure defined in **<sundev/msio.h>**:

```
typedef struct {
int     jitter_thresh;
int     speed_law;
int     speed_limit;
}       Ms_parms;
```

*jitter_thresh* is the "jitter threshold" of the mouse. Motions of fewer than *jitter_thresh* units along both axes that occur in less than 1/12 second are treated as "jitter" and ignored. Thus, if the mouse moves fewer than *jitter_thresh units* and then moves back to its original position in less than 1/12 of a second, the motion is considered to be "noise" and ignored. If it moves fewer than *jitter_thresh* units and continues to move so that it has not returned to its original position after 1/12 of a second, the motion is considered to be real and is reported.

*speed_law* indicates whether extremely large motions are to be ignored. If it is 1, a "speed limit" is applied to mouse motions; motions along either axis of more than *speed_limit* units are discarded.

Note: these parameters are global; if they are set for any mouse on a workstation, they apply to any other mice attached to that workstation as well.

**VUIDSFORMAT**
**VUIDGFORMAT**
**VUIDSADDR**
**VUIDGADDR** These are standard *Virtual User Input Device ioctl*s. See *SunView 1 System Programmer's Guide* for a description of their operation.

**MSIOGETPARMS** The argument is a pointer to a **Ms_parms**. The current mouse parameters are stored in that structure.

**MSIOSETPARMS** The argument is a pointer to a **ms_parms**. The current mouse parameters are set from the values in that structure.

**SEE ALSO**

**mouse**(4S), **termios**(4), **win**(4S), **zs**(4S)

*SunView 1 System Programmer's Guide*

**NAME**

mti – Systech MTI-800/1600 multi-terminal interface

**CONFIG — SUN-3 SYSTEM**

**device mti0 at vme16d16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88**
**device mti1 at vme16d16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89**
**device mti2 at vme16d16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a**
**device mti3 at vme16d16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b**

**CONFIG — SUN-2 SYSTEM**

**device mti0 at mbio ? csr 0x620 flags 0xffff priority 4**
**device mti1 at mbio ? csr 0x640 flags 0xffff priority 4**
**device mti2 at mbio ? csr 0x660 flags 0xffff priority 4**
**device mti3 at mbio ? csr 0x680 flags 0xffff priority 4**
**device mti0 at vme16 ? csr 0x620 flags 0xffff priority 4 vector mtiintr 0x88**
**device mti1 at vme16 ? csr 0x640 flags 0xffff priority 4 vector mtiintr 0x89**
**device mti2 at vme16 ? csr 0x660 flags 0xffff priority 4 vector mtiintr 0x8a**
**device mti3 at vme16 ? csr 0x680 flags 0xffff priority 4 vector mtiintr 0x8b**

**SYNOPSIS**

**#include <fcntl.h>**
**#include <sys/termios.h>**
**open("/dev/tty*xy*", mode);**
**open("/dev/ttyd*n*", mode);**
**open("/dev/cua*n*", mode);**

**DESCRIPTION**

The Systech MTI card provides 8 (MTI-800) or 16 (MTI-1600) serial communication lines with modem control. Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termios** structure are performed by the **mti** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x0004** in the specification of **mti0** would treat line **/dev/tty02** in this way.

Minor device numbers in the range 0 – 63 correspond directly to the normal tty lines and are named **/dev/tty***XY*, where *X* is the physical board number (0 – 3), and *Y* is the line number on the board as a single hexadecimal digit. (Thus the first line on the first board is **/dev/tty00**, and the sixteenth line on the third board is **/dev/tty2f**.)

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 191 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 63 and is conventionally renamed **/dev/ttyd***n*, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua***n*, where *n* is the number of the dial-in line.

The **/dev/cua***n* lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua***n* line is opened, the corresponding tty line can not be opened until the **/dev/cua***n* line is closed; a blocking open will wait until the **/dev/cua***n* line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd***n* line has been opened successfully (usually only when carrier is

recognized on the modem) the corresponding **/dev/cua***n* line can not be opened. This allows a modem to be attached to e.g. **/dev/ttyd0** (renamed from **/dev/tty00**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

WIRING

The Systech requires the CTS modem control signal to operate. If the device does not supply CTS then RTS should be jumpered to CTS at the distribution panel (short pins 4 to 5). Also, the CD (carrier detect) line does not work properly. When connecting a modem, the modem's CD line should be wired to DSR, which the software will treat as carrier detect.

IOCTLS

The standard set of **termio ioctl**( ) calls are supported by **mti**.

The state of the **CRTSCTS** flag in the **c_cflag** word has no effect; no output will be generated unless CTS is high. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl**( ) calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided; however, as described above, the DSR line is treated as CD and the CD line is ignored.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed. The baud rates **B200** and **B38400** are not supported by the hardware; **B200** selects 2000 baud, and **B38400** selects 7200 baud.

ERRORS

An **open**( ) will fail if:

| | |
|---|---|
| ENXIO | The unit being opened does not exist. |
| EBUSY | The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open. |
| EBUSY | The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl**( ) call. |
| EINTR | The open was interrupted by the delivery of a signal. |

FILES

| | |
|---|---|
| **/dev/tty[0-3][0-9a-f]** | hardwired tty lines |
| **/dev/ttyd[0-9a-f]** | dialin tty lines |
| **/dev/cua[0-9a-f]** | dialout tty lines |

SEE ALSO

**tip**(1C), **uucp**(1C), **mcp**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **zs**(4S)

DIAGNOSTICS

Most of these diagnostics "should never happen" and their occurrence usually indicates problems elsewhere in the system.

**mti***n*, *n*: **silo overflow.**
More than 512 characters have been received by the mti hardware without being read by the software. Extremely unlikely to occur.

**mti***n*: **read error code <***n***>. Probable hardware fault**
The **mti** returned the indicated error code. See the MTI manual.

**mti***n*: **DMA output error.**
The **mti** encountered an error while trying to do DMA output.

**mti***n*: **impossible response** *n*.
The **mti** returned an error it could not understand.

## NAME

mtio – general magnetic tape interface

## SYNOPSIS

**#include <sys/ioctl.h>**
**#include <sys/mtio.h>**

## DESCRIPTION

Both 1/2'' and 1/4'' magnetic tape drives share the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

The ''cooked'' magnetic tape device files read and write magnetic tape in 2048 byte blocks (the 2048 is actually **BLKDEV_IOSIZE** in **<sys/param.h>**). The name of such a device file might be **/dev/mt0**. The final component of the name is composed of a name that represents the type of device the file refers to, and the unit number of that device.

These files are rewound when closed; the ''no-rewind'' versions of these files are not. The name of ''no-rewind'' device files include the letter **n** at the beginning of the final component of the name; the ''no-rewind'' version of **/dev/mt0** would be **/dev/nmt0**. When a 1/2'' tape file, open for writing or just written, is closed, two tape marks are written; if the tape is not to be rewound it is positioned with the head between the two tapemarks. When a 1/4'' tape file, (due to a bug, only if) just written, is closed, only one end of file mark is written because of the inability to overwrite data on a 1/4'' tape; see below.

The files discussed above are useful when you want to access the tape in a way compatible with ordinary files. This interface requires that all blocks be 2048 bytes long, and does not permit special operations (such as spacing the tape forward or backward) to be performed. When using foreign tapes, and especially when reading or writing long records, the ''raw'' interface is appropriate. The name of ''raw'' device files include the letter **r** before the device type; the ''raw'' version of **/dev/mt0** would be **/dev/rmt0**, and the ''raw'' version of **/dev/nmt0** would be **/dev/nrmt0**. Each **read**(2V) or **write**(2V) call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, provided it is no greater than the buffer size. In ''raw'' tape I/O, seeks are ignored. A zero byte count is returned when a tape mark is read, but another read will fetch the first record of the new tape file.

1/4'' tapes are not able to back up and always write fixed sized blocks. Since they cannot back up, they cannot support backward space file and backward space record. Since they always write fixed sized blocks, the size of transfers using the ''raw'' interface must be a multiple of the underlying blocksize, usually 512 bytes.

1/4'' tapes also have an unusual tape format. They have parallel tracks, but only record information on one track at a time, switching to another track near the physical end of the medium. They erase all the tracks at once while writing the first track. Therefore, they cannot, in general, overwrite previously written data. If the old data were not on the first track, it would not be erased before being overwritten, and the result would be unreadable.

A number of additional ioctl operations are available on ''raw'' devices. The following definitions are from **<sys/mtio.h>**:

```
/*
 * Structures and definitions for mag tape I/O control commands
 */

/* structure for MTIOCTOP - mag tape op command */
struct  mtop    {
        short   mt_op;                  /* operations defined below */
        daddr_t mt_count;               /* how many of them */
};
```

```
/* operations */
#define  MTWEOF      0          /* write an end-of-file record */
#define  MTFSF       1          /* forward space file */
#define  MTBSF       2          /* backward space file */
#define  MTFSR       3          /* forward space record */
#define  MTBSR       4          /* backward space record */
#define  MTREW       5          /* rewind */
#define  MTOFFL      6          /* rewind and put the drive offline */
#define  MTNOP       7          /* no operation, sets status only */
#define  MTRETEN     8          /* retension the tape */
#define  MTERASE     9          /* erase the entire tape */
#define  MTEOM       10         /* position to end of media (SCSI only) */

/* structure for MTIOCGET - mag tape get status command */

struct   mtget   {
         short   mt_type;       /* type of magtape device */

/* the following two registers are grossly device dependent */
         short   mt_dsreg;      /* "drive status" register */
         short   mt_erreg;      /* "error" register */

/* end device-dependent registers */
         short   mt_resid;      /* residual count */

/* the following two are not yet implemented */
         daddr_t mt_fileno;     /* file number of current position */
         daddr_t mt_blkno;      /* block number of current position */

/* end not yet implemented */
};

/*
 * Constants for mt_type byte
 */
#define  MT_ISTS     0x01       /* vax: unibus ts-11 */
#define  MT_ISHT     0x02       /* vax: massbus tu77, etc */
#define  MT_ISTM     0x03       /* vax: unibus tm-11 */
#define  MT_ISMT     0x04       /* vax: massbus tu78 */
#define  MT_ISUT     0x05       /* vax: unibus gcr */
#define  MT_ISCPC    0x06       /* sun: Multibus tapemaster */
#define  MT_ISAR     0x07       /* sun: Multibus archive */
#define  MT_ISSC     0x08       /* sun: SCSI archive */
#define  MT_ISXY     0x09       /* sun: Xylogics 472 */
#define  MT_ISSYSGEN 0x0a       /* sun: SCSI Sysgen */
#define  MT_ISMT02   0x0b       /* sun: SCSI Emulex MT02 */
#define  MT_ISCCS    0x0c       /* sun: SCSI generic (unknown) CCS */

/* mag tape io control commands */
#define  MTIOCTOP    _IOW(m, 1, struct mtop) /* do a mag tape op */
#define  MTIOCGET    _IOR(m, 2, struct mtget) /* get tape status */
#ifndef KERNEL
#define  DEFTAPE     "/dev/rmt12"
#endif
```

**SEE ALSO**

**mt**(1), **tar**(1), **read**(2V), **write**(2V), **ar**(4S), **tm**(4S), **st**(4S), **xt**(4S)

## NAME

nfs, NFS – network file system

## CONFIG

**options NFS**

## DESCRIPTION

The Network File System, or NFS, allows a client workstation to perform transparent file access over the network. Using it, a client workstation can operate on files that reside on a variety of servers, server architectures and across a variety of operating systems. Client file access calls are converted to NFS protocol requests, and are sent to the server system over the network. The server receives the request, performs the actual file system operation, and sends a response back to the client.

The Network File System operates in a stateless fashion using remote procedure (RPC) calls built on top of external data representation (XDR) protocol. These protocols are documented in *Network Programming* The RPC protocol provides for version and authentication parameters to be exchanged for security over the network.

A server can grant access to a specific filesystem to certain clients by adding an entry for that filesystem to the server's **/etc/exports** file.

A client gains access to that filesystem with the **mount**(2) system call, which requests a file handle for the filesystem itself. Once the filesystem is mounted by the client, the server issues a file handle to the client for each file (or directory) the client accesses. If the file is somehow removed on the server side, the file handle becomes stale (dissociated with a known file).

A server may also be a client with respect to filesystems it has mounted over the network, but its clients cannot gain access to those filesystems. Instead, the client must mount a filesystem directly from the server on which it resides.

The user ID and group ID mappings must be the same between client and server. However, the server maps uid 0 (the super-user) to uid –2 before performing access checks for a client. This inhibits super-user privileges on remote filesystems.

NFS-related routines and structure definitions are described in *Network Programming*.

## ERRORS

Generally physical disk I/O errors detected at the server are returned to the client for action. If the server is down or inaccessible, the client will see the console message:

**NFS: file server not responding: still trying.**

The client continues (forever) to resend the request until it receives an acknowledgement from the server. This means the server can crash or power down, and come back up, without any special action required by the client. It also means the client process requesting the I/O will block and remain insensitive to signals, sleeping inside the kernel at **PRIBIO**.

## FILES

**/etc/exports**

## SEE ALSO

**mount**(2), **exports**(5), **fstab**(5), **fstab**(5), **mount**(8), **nfsd**(8)

*Network Programming*

## NAME

nit – Network Interface Tap

## CONFIG

**pseudo-device clone**
**pseudo-device snit**
**pseudo-device pf**
**pseudo-device nbuf**

## SYNOPSIS

```
#include <sys/file.h>
#include <sys/ioctl.h>
#include <net/nit_pf.h>
#include <net/nit_buf.h>

        fd = open("/dev/nit", mode);
        ioctl(fd, I_PUSH, "pf");
        ioctl(fd, I_PUSH, "nbuf");
```

## DESCRIPTION

NIT (the Network Interface Tap) is a facility composed of several STREAMS modules and drivers. These components collectively provide facilities for constructing applications that require link-level network access. Examples of such applications include **rarpd**(8C), which is a user-level implementation of the Reverse ARP protocol, and **etherfind**(8C), which is a network monitoring and trouble-shooting program.

NIT consists of several components that are summarized below. See their Reference Manual entries for detailed information about their specification and operation.

**nit_if**(4M) This component is a STREAMS device driver that interacts directly with the system's Ethernet drivers. After opening an instance of this device it must be bound to a specific Ethernet interface before becoming usable. Subsequently, **nit_if** transcribes packets arriving on the interface to the read side of its associated stream and delivers messages reaching it on the write side of its stream to the raw packet output code for transmission over the interface.

**nit_pf**(4M) This module provides packet-filtering services, allowing uninteresting incoming packets to be discarded with minimal loss of efficiency. It passes through unaltered all outgoing messages (those on the stream's write side).

**nit_buf**(4M) This module buffers incoming messages into larger aggregates, thereby reducing the overhead incurred by repeated **read**(2V) system calls.

NIT clients mix and match these components, based on their particular requirements. For example, the reverse ARP daemon concerns itself only with packets of a specific type and deals with low traffic volumes. Thus, it uses **nit_if** for access to the network and **nit_pf** to filter out all incoming packets except reverse ARP packets, but omits the **nit_buf** buffering module since traffic isn't high enough to justify the additional complexity of unpacking buffered packets. On the other hand, the **etherd**(8C) program, which collects Ethernet statistics for **traffic**(1C) to display, must examine every packet on the network. Therefore, it omits the **nit_if** module, since there's nothing it wishes to screen out, and includes the **nit_buf** module, since most networks have very heavy aggregate packet traffic.

## EXAMPLES

The following code fragments outline how to program against parts of the NIT interface. For the sake of brevity, all error-handling code has been elided.

**initdevice** comes from **etherfind** and sets up its input stream configuration.

```
initdevice(if_flags, snaplen, chunksize)
        u_long    if_flags,
                  snaplen,
                  chunksize;
{
        struct strioctl        si;
```

```
        struct ifreq        ifr;
        struct timeval      timeout;

        if_fd = open(NIT_DEV, O_RDONLY);

        /* Arrange to get discrete messages from the stream. */
        ioctl(if_fd, I_SRDOPT, (char *)RMSGD);

        si.ic_timout = INFTIM;

        /* Push and configure the buffering module. */
        ioctl(if_fd, I_PUSH, "nbuf");

        timeout.tv_sec = 1;
        timeout.tv_usec = 0;
        si.ic_cmd = NIOCSTIME;
        si.ic_len = sizeof timeout;
        si.ic_dp = (char *)&timeout;
        ioctl(if_fd, I_STR, (char *)&si);

        si.ic_cmd = NIOCSCHUNK;
        si.ic_len = sizeof chunksize;
        si.ic_dp = (char *)&chunksize;
        ioctl(if_fd, I_STR, (char *)&si);

        /* Configure the nit device, binding it to the proper
          underlying interface, setting the snapshot length,
          and setting nit_if-level flags. */
        strncpy(ifr.ifr_name, device, sizeof ifr.ifr_name);
        ifr.ifr_name[sizeof ifr.ifr_name - 1] = '  ';
        si.ic_cmd = NIOCBIND;
        si.ic_len = sizeof ifr;
        si.ic_dp = (char *)&ifr;
        ioctl(if_fd, I_STR, (char *)&si);

        if (snaplen > 0) {
                si.ic_cmd = NIOCSSNAP;
                si.ic_len = sizeof snaplen;
                si.ic_dp = (char *)&snaplen;
                ioctl(if_fd, I_STR, (char *)&si);
        }

        if (if_flags != 0) {
                si.ic_cmd = NIOCSFLAGS;
                si.ic_len = sizeof if_flags;
                si.ic_dp = (char *)&if_flags;
                ioctl(if_fd, I_STR, (char *)&si);
        }

        /* Flush the read queue, to get rid of anything that accumulated
          before the device reached its final configuration. */
        ioctl(if_fd, I_FLUSH, (char *)FLUSHR);
}
```

Here is the skeleton of the packet reading loop from **etherfind**. It illustrates how to cope with dismantling the headers the various NIT components glue on.

```
        while ((cc = read(if_fd, buf, chunksize)) >= 0) {
                register u_char       *bp = buf,
                                      *bufstop = buf + cc;

                /* Loop through each message in the chunk. */
                while (bp < bufstop) {
                        register u_char                 *cp = bp;
                        struct nit_bufhdr     *hdrp;
```

```
                struct timeval          *tvp = NULL;
                u_long                  drops = 0;
                u_long                  pktlen;

                /* Extract information from the successive objects
                   embedded in the current message. Which ones we
                   have depends on how we set up the stream (and
                   therefore on what command line flags were set).

                   If snaplen is positive then the packet was truncated
                   before the buffering module saw it, so we must
                   obtain its length from the nit_if-level nit_iflen
                   header. Otherwise the value in *hdrp suffices. */
                hdrp = (struct nit_bufhdr *)cp;
                cp += sizeof *hdrp;
                if (tflag) {
                        struct nit_iftime       *ntp;

                        ntp = (struct nit_iftime *)cp;
                        cp += sizeof *ntp;

                        tvp = &ntp->nh_timestamp;
                }
                if (dflag) {
                        struct nit_ifdrops      *ndp;

                        ndp = (struct nit_ifdrops *)cp;
                        cp += sizeof *ndp;

                        drops = ndp->nh_drops;
                }
                if (snaplen > 0) {
                        struct nit_iflen        *nlp;

                        nlp = (struct nit_iflen *)cp;
                        cp += sizeof *nlp;

                        pktlen = nlp->nh_pktlen;
                }
                else
                        pktlen = hdrp->nhb_msglen;

                sp = (struct sample *)cp;
                bp += hdrp->nhb_totlen;

                /* Process the packet. */
        }
}
```

**FILES**

**/dev/nit** clone device instance referring to **nit_if**

**SEE ALSO**

**traffic**(1C), **read**(2V), **nit_if**(4M), **nit_pf**(4M), **nit_buf**(4M), **etherd**(8C), **etherfind**(8C), **rarpd**(8C)

## NAME

nit_buf – STREAMS NIT buffering module

## CONFIG

**pseudo-device nbuf**

## SYNOPSIS

```
#include <sys/ioctl.h>
#include <net/nit_buf.h>
ioctl(fd, I_PUSH, "nbuf");
```

## DESCRIPTION

**nit_buf** is a STREAMS module that buffers incoming messages, thereby reducing the number of system calls and associated overhead required to read and process them. Although designed to be used in conjunction with the other components of NIT (see **nit**(4P)), **nit_buf** is a general-purpose module and can be used anywhere STREAMS input buffering is required.

### Read-side Behavior

**nit_buf** collects incoming **M_DATA** and **M_PROTO** messages into *chunks*, passing each chunk upward when either the chunk becomes full or the current read timeout expires. When a message arrives, it is processed in two steps. First, the message is prepared for inclusion in a chunk, and then it is added to the current chunk. The following paragraphs discuss each step in turn.

Upon receiving a message from below, **nit_buf** immediately converts all leading **M_PROTO** blocks in the message to **M_DATA** blocks, altering only the message type field and leaving the contents alone. It then prepends a header to the converted message. This header is defined as follows.

```
struct nit_bufhdr {
        u_int   nhb_msglen;
        u_int   nhb_totlen;
};
```

The first field of this header gives the length in bytes of the converted message. The second field gives the distance in bytes from the start of the message in the current chunk (described below) to the start of the next message in the chunk; the value reflects any padding necessary to insure correct data alignment for the host machine and includes the length of the header itself.

After preparing a message, **nit_buf** attempts to add it to the end of the current chunk, using the chunk size and timeout values to govern the addition. (The chunk size and timeout values are set and inspected using the **ioctl** calls described below.) If adding the new message would make the current chunk grow larger than the chunk size, **nit_buf** closes off the current chunk, passing it up to the next module in line, and starts a new chunk, seeding it with a zero-length message. If adding the message would still make the current chunk overflow, the module passes it upward in an over-size chunk of its own. Otherwise, the module concatenates the message to the end of the current chunk.

To ensure that messages do not languish forever in an accumulating chunk, **nit_buf** maintains a read timeout. Whenever this timeout expires, the module closes off the current chunk, regardless of its length, and passes it upward; if no incoming messages have arrived, the chunk passed upward will have zero length. Whenever the module passes a chunk upward, it restarts the timeout period. These two rules insure that **nit_buf** minimizes the number of chunks it produces during periods of intense message activity and that it periodically disposes of all messages during slack intervals.

**nit_buf** handles other message types as follows. Upon receiving an **M_FLUSH** message specifying that the read queue be flushed, the module does so, clearing the currently accumulating chunk as well, and passes the message on to the module or driver above. It passes all other messages through unaltered to its upper neighbor.

### Write-side Behavior

**nit_buf** intercepts **M_IOCTL** messages for the *ioctl*s described below. Upon receiving an **M_FLUSH** message specifying that the write queue be flushed, the module does so and passes the message on to the module or driver below. The module passes all other messages through unaltered to its lower neighbor.

## IOCTLS

**nit_buf** responds to the following *ioctl*s.

**NIOCSTIME** Set the read timeout value to the value referred to by the *struct timeval* pointer given as argument. Setting the timeout value to zero has the side-effect of forcing the chunk size to zero as well, so that the module will pass all incoming messages upward immediately upon arrival.

**NIOCGTIME** Return the read timeout in the *struct timeval* pointed to by the argument. If the timeout has been cleared with the **NIOCCTIME** *ioctl*, return with an ERANGE error.

**NIOCCTIME** Clear the read timeout, effectively setting its value to infinity.

**NIOCSCHUNK** Set the chunk size to the value referred to by the *u_int* pointer given as argument.

**NIOCGCHUNK** Return the chunk size in the *u_int* pointed to by the argument.

## CAVEAT

The module name "nbuf" used in the system configuration file and as argument to the **I_PUSH ioctl** is provisional and subject to change.

## SEE ALSO

**nit**(4P), **nit_if**(4M), **nit_pf**(4M)

## NAME

nit_if – STREAMS NIT device interface module

## CONFIG

**pseudo-device snit**

## SYNOPSIS

**#include <sys/file.h>**
**open("/dev/nit", mode);**

## DESCRIPTION

**nit_if** is a STREAMS pseudo-device driver that provides STREAMS access to network interfaces. It is designed to be used in conjunction with the other components of NIT (see **nit**(4P)), but can be used by itself as a raw STREAMS network interface.

**nit_if** is an exclusive-open device that is intended to be opened indirectly through the clone device; **/dev/nit** is a suitable instance of the clone device. Before the stream resulting from opening an instance of **nit_if** may be used to read or write packets, it must first be bound to a specific network interface, using the **NIOCSBIND** ioctl described below.

### Read-side Behavior

**nit_if** copies leading prefixes of selected packets from its associated network interface and passes them up the stream. If the **NI_PROMISC** flag is set, it passes along all packets; otherwise it passes along only packets addressed to the underlying interface.

The amount of data copied from a given packet depends on the current *snapshot length*, which is set with the **NIOCSSNAP** ioctl described below.

Before passing each packet prefix upward, **nit_if** optionally prepends one or more headers, as controlled by the state of the flag bits set with the **NIOCSFLAGS** *ioctl*. The driver collects headers into **M_PROTO** message blocks, with the headers guaranteed to be completely contained in a single message block, whereas the packet itself goes into one or more **M_DATA** message blocks.

### Write-side Behavior

**nit_if** accepts packets from the module above it in the stream and relays them to the associated network interface for transmission. Packets must be formatted with the destination address in a leading **M_PROTO** message block, followed by the packet itself, complete with link-level header, in a sequence of **M_DATA** message blocks. The destination address must be expressed as a '**struct sockaddr**' whose *sa_family* field is **AF_UNSPEC** and whose *sa_data* field is a copy of the link-level header. (See **<sys/socket.h>** for the definition of this structure.)

**nit_if** processes **M_IOCTL** messages as described below. Upon receiving an **M_FLUSH** message specifying that the write queue be flushed, **nit_if** does so and transfers the message to the read side of the stream. It discards all other messages.

## IOCTLS

**nit_if** responds to the following *ioctl*s, as defined in **<net/nit_if.h>**. It generates an **M_IOCNAK** message for all others, returning this message to the invoker along the read side of the stream.

**SIOCGIFADDR** **nit_if** passes this ioctl on to the underlying interface's driver and returns its response in a '**struct ifreq**' instance, as defined in **<net/if.h>**. (See the description of this ioctl in **if**(4N) for more details.)

**NIOCBIND** This ioctl attaches the stream represented by its first argument to the network interface designated by its third argument, which should be a pointer to an *ifreq* structure whose *ifr_name* field names the desired interface. See **<net/if.h>** for the definition of this structure.

**NIOCSSNAP** Set the current snapshot length to the value given in the *u_long* pointed to by the *ioctl*'s final argument. **nit_if** interprets a snapshot length value of zero as meaning infinity, so that it will copy all selected packets in their entirety. It constrains

positive snapshot lengths to be at least the length of an Ethernet header, so that it will pass at least the link-level header of all selected packets to its upstream neighbor.

**NIOCGSNAP** Returns the current snapshot length for this device instance in the *u_long* pointed to by the *ioctl*'s final argument.

**NIOCSFLAGS** **nit_if** recognizes the following flag bits, which must be given in the *u_long* pointed to by the *ioctl*'s final argument. This set may be augmented in future releases. All but the **NI_PROMISC** bit control the addition of headers that precede the packet body. These headers appear in the order given below, with the last-mentioned enabled header adjacent to the packet body.

**NI_PROMISC** Requests that the underlying interface be set into promiscuous mode and that all packets that the interface receives be passed up through the stream. **nit_if** only honors this bit for the super-user.

**NI_TIMESTAMP** Prepend to each selected packet a header containing the packet arrival time expressed as a '**struct timeval**'.

**NI_DROPS** Prepend to each selected packet a header containing the cumulative number of packets that this instance of **nit_if** has dropped because of flow control requirements or resource exhaustion. The header value is expressed as a *u_long*. Note: it accounts only for events occurring within **nit_if**, and does not count packets dropped at the network interface level or by upstream modules.

**NI_LEN** Prepend to each selected packet a header containing the packet's original length (including link-level header), as it was before being trimmed to the snapshot length. The header value is expressed as a *u_long*.

**NIOCGFLAGS** Returns the current state of the flag bits for this device instance in the *u_long* pointed to by the *ioctl*'s final argument.

FILES

**/dev/nit** clone device instance referring to **nit_if** device

**<net/nit_if.h>** header file containing definitions for the *ioctl*s and packet headers described above.

SEE ALSO

**clone**(4), **nit**(4P), **nit_buf**(4M), **nit_pf**(4M)

## NAME

nif_pf – STREAMS NIT packet filtering module

## CONFIG

**pseudo-device pf**

## SYNOPSIS

**#include <sys/ioctl.h>**
**#include <net/nit_pf.h>**

**ioctl(*fd*, I_PUSH, "pf");**

## DESCRIPTION

**nit_pf** is a STREAMS module that subjects messages arriving on its read queue to a packet filter and passes only those messages that the filter accepts on to its upstream neighbor. Such filtering can be very useful for user-level protocol implementations and for networking monitoring programs that wish to view only specific types of events.

### Read-side Behavior

**nit_pf** applies the current packet filter to all **M_DATA** and **M_PROTO** messages arriving on its read queue. The module prepares these messages for examination by first skipping over all leading **M_PROTO** message blocks to arrive at the beginning of the message's data portion. If there is no data portion, **nit_pf** accepts the message and passes it along to its upstream neighbor. Otherwise, the module ensures that the part of the message's data that the packet filter might examine lies in contiguous memory, calling the *pullupmsg* utility routine if necessary to force contiguity. (Note: this action destroys any sharing relationships that the subject message might have had with other messages.) Finally, it applies the packet filter to the message's data, passing the entire message upstream to the next module if the filter accepts, and discarding the message otherwise. See PACKET FILTERS below for details on how the filter works.

If there is no packet filter yet in effect, the module acts as if the filter exists but does nothing, implying that all incoming messages are accepted. IOCTLS below describes how to associate a packet filter with an instance of **nit_pf**.

**nit_pf** handles other message types as follows. Upon receiving an **M_FLUSH** message specifying that the read queue be flushed, the module does so, and passes the message on to its upstream neighbor. It passes all other messages through unaltered to its upper neighbor.

### Write-side Behavior

**nit_pf** intercepts **M_IOCTL** messages for the *ioctl* described below. Upon receiving an **M_FLUSH** message specifying that the write queue be flushed, the module does so and passes the message on to the module or driver below. The module passes all other messages through unaltered to its lower neighbor.

## IOCTLS

**nit_pf** responds to the following *ioctl*.

NIOCSETF This *ioctl* directs the module to replace its current packet filter, if any, with the filter specified by the '**struct packetfilt**' pointer named by its final argument. This structure is defined in **<net/packetfilt.h>** as

```
struct packetfilt {
    u_char   Pf_Priority;    /* priority of filter */
    u_char   Pf_FilterLen;   /* # of cmds in list */
    u_short  Pf_Filter[ENMAXFILTERS];
                             /* filter command list */
};
```

The *Pf_Priority* field is included only for compatibility with other packet filter implementations and is otherwise ignored. The packet filter itself is specified in the *Pf_Filter* array as a sequence of two-byte commands, with the *Pf_FilterLen* field giving the number of commands in the sequence. This implementation restricts the maximum number of commands in a filter (ENMAXFILTERS) to 40. The next section describes the available commands and their semantics.

## PACKET FILTERS

A packet filter consists of the filter command list length (in units of *u_shorts*), and the filter command list itself. (The priority field mentioned above is ignored in this implementation.) Each filter command list specifies a sequence of actions that operate on an internal stack of *u_shorts* ("shortwords"). Each shortword of the command list specifies one of the actions ENF_PUSHLIT, ENF_PUSHZERO, or ENF_PUSHWORD+*n*, which respectively push the next shortword of the command list, zero, or shortword *n* of the subject message on the stack, and a binary operator from the set { ENF_EQ, ENF_NEQ, ENF_LT, ENF_LE, ENF_GT, ENF_GE, ENF_AND, ENF_OR, ENF_XOR } which then operates on the top two elements of the stack and replaces them with its result. When both an action and operator are specified in the same shortword, the action is performed followed by the operation.

The binary operator can also be from the set { ENF_COR, ENF_CAND, ENF_CNOR, ENF_CNAND }. These are "short-circuit" operators, in that they terminate the execution of the filter immediately if the condition they are checking for is found, and continue otherwise. All pop two elements from the stack and compare them for equality; ENF_CAND returns false if the result is false; ENF_COR returns true if the result is true; ENF_CNAND returns true if the result is false; ENF_CNOR returns false if the result is true. Unlike the other binary operators, these four do not leave a result on the stack, even if they continue.

The short-circuit operators should be used when possible, to reduce the amount of time spent evaluating filters. When they are used, you should also arrange the order of the tests so that the filter will succeed or fail as soon as possible; for example, checking the IP destination field of a UDP packet is more likely to indicate failure than the packet type field.

The special action ENF_NOPUSH and the special operator ENF_NOP can be used to only perform the binary operation or to only push a value on the stack. Since both are (conveniently) defined to be zero, indicating only an action actually specifies the action followed by ENF_NOP, and indicating only an operation actually specifies ENF_NOPUSH followed by the operation.

After executing the filter command list, a non-zero value (true) left on top of the stack (or an empty stack) causes the incoming packet to be accepted and a zero value (false) causes the packet to be rejected. (If the filter exits as the result of a short-circuit operator, the top-of-stack value is ignored.) Specifying an undefined operation or action in the command list or performing an illegal operation or action (such as pushing a shortword offset past the end of the packet or executing a binary operator with fewer than two shortwords on the stack) causes a filter to reject the packet.

## EXAMPLES

The reverse ARP daemon program (**rarpd**(8C)) uses code similar to the following fragment to construct a filter that rejects all but RARP packets. That is, is accepts only packets whose Ethernet type field has the value ETHERTYPE_REVARP.

```
struct ether_header eh;          /* used only for offset values */
struct packetfilt pf;
register u_short *fwp = pf.Pf_Filter;
u_short offset;

/*
 * Set up filter. Offset is the displacement of the Ethernet
 * type field from the beginning of the packet in units of
 * u_shorts.
 */
```

```
        offset = ((u_int) &eh.ether_type - (u_int) &eh.ether_dhost) /
sizeof (u_short);
        *fwp++ = ENF_PUSHWORD + offset;
        *fwp++ = ENF_PUSHLIT;
        *fwp++ = htons(ETHERTYPE_REVARP);
        *fwp++ = ENF_EQ;
        pf.Pf_FilterLen = fwp - &pf.Pf_Filter[0];
```

This filter can be abbreviated by taking advantage of the ability to combine actions and operations:

```
        ...
        *fwp++ = ENF_PUSHWORD + offset;
        *fwp++ = ENF_PUSHLIT | ENF_EQ;
        *fwp++ = htons(ETHERTYPE_REVARP);
        ...
```

## WARNINGS

The module name '**pf**' used in the system configuration file and as argument to the **I_PUSH** *ioctl* is provisional and subject to change.

The *Pf_Priority* field of the *packetfilt* structure is likely to be removed.

## SEE ALSO

**inet**(4F), **nit**(4P), **nit_buf**(4M), **nit_if**(4M)

**NAME**

null – data sink

**CONFIG**

None; included with standard system.

**SYNOPSIS**

**#include <fcntl.h>**

**open("/dev/null",** *mode*);

**DESCRIPTION**

Data written on the **null** special file is discarded.

Reads from the **null** special file always return an end-of-file indication.

**FILES**

**/dev/null**

NAME

pp – Centronics-compatible parallel printer port

CONFIG

**device pp0 at obio ? csr**

AVAILABILITY

Sun386i systems only.

DESCRIPTION

This device driver provides an interface to the Sun386i system's on-board Centronics-compatible parallel printer port. It supports most standard PC printers with Centronics interfaces.

FILES

**/dev/pp0**

DIAGNOSTICS

**pp*: printer not online**
**pp*: printer out of paper**

NAME

pty – pseudo-terminal driver

CONFIG

**pseudo-device pty*n***

SYNOPSIS

**#include <fcntl.h>**
**#include <sys/termios.h>**
**open("/dev/ttyp*n*", mode);**
**open("/dev/ptyp*n*", mode);**

DESCRIPTION

The **pty** driver provides support for a pair of devices collectively known as a *pseudo-terminal*. The two devices comprising a pseudo-terminal are known as a *controller* and a *slave*. The slave device distinguishes between the **B0** baud rate and other baud rates specified in the **c_cflag** word of the **termios** structure, and the **CLOCAL** flag in that word. It does not support any of the other **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the **IGNBRK, IGNPAR, PARMRK,** or **INPCK** flags in the **c_iflag** word of the **termios** structure, as these functions apply only to asynchronous serial ports. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a slave device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Instead of having a hardware interface and associated hardware that supports the terminal functions, the functions are implemented by another process manipulating the controller device of the pseudo-terminal.

The controller and the slave devices of the pseudo-terminal are tightly connected. Any data written on the controller device is given to the slave device as input, as though it had been received from a hardware interface. Any data written on the slave terminal can be read from the controller device (rather than being transmitted from a UART).

In configuring, if no optional "count" is given in the specification, 16 pseudo-terminal pairs are configured.

IOCTLS

The standard set of **termio ioctl**s are supported by the slave device. None of the bits in the **c_cflag** word have any effect on the pseudo-terminal, except that if the baud rate is set to **B0**, it will appear to the process on the controller device as if the last process on the slave device had closed the line; thus, setting the baud rate to **B0** has the effect of "hanging up" the pseudo-terminal, just as it has the effect of "hanging up" a real terminal.

There is no notion of "parity" on a pseudo-terminal, so none of the flags in the **c_iflag** word that control the processing of parity errors have any effect. Similarly, there is no notion of a "break", so none of the flags that control the processing of breaks, and none of the **ioctl**s that generate breaks, have any effect.

Input flow control is automatically performed; a process that attempts to write to the controller device will be blocked if too much unconsumed data is buffered on the slave device. The input flow control provided by the **IXOFF** flag in the **c_iflag** word is not supported.

The delays specified in the **c_oflag** word are not supported.

As there are no modems involved in a pseudo-terminal, the **ioctl**s that return or alter the state of modem control lines are silently ignored.

On Sun systems, an additional **ioctl** is provided:

TIOCCONS

The argument is ignored. All output that would normally be sent to the console (either from programs writing to **/dev/console** or from kernel printouts) is redirected so that it is written to the pseudo-terminal instead.

A few special **ioctl**s are provided on the controller devices of pseudo-terminals to provide the functionality needed by applications programs to emulate real hardware interfaces:

**TIOCSTOP**
The argument is ignored. Output to the pseudo-terminal is suspended, as if a **STOP** character had been typed.

**TIOCSTART**
The argument is ignored. Output to the pseudo-terminal is restarted, as if a **START** character had been typed.

**TIOCPKT**
The argument is a pointer to an **int**. If the value of the **int** is non-zero, *packet* mode is enabled; if the value of the **int** is zero, packet mode is disabled. When a pseudo-terminal is in packet mode, each subsequent **read**(2V) from the controller device will return data written on the slave device preceded by a zero byte (symbolically defined as **TIOCPKT_DATA**), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

**TIOCPKT_FLUSHREAD**
whenever the read queue for the terminal is flushed.

**TIOCPKT_FLUSHWRITE**
whenever the write queue for the terminal is flushed.

**TIOCPKT_STOP** whenever output to the terminal is stopped using ^S.

**TIOCPKT_START** whenever output to the terminal is restarted.

**TIOCPKT_DOSTOP** whenever XON/XOFF flow control is enabled after being disabled; it is considered "enabled" when the **IXON** flag in the **c_iflag** word is set, the **VSTOP** member of the **c_cc** array is ^S and the **VSTART** member of the **c_cc** array is ^Q.

**TIOCPKT_NOSTOP** whenever XON/XOFF flow control is disabled after being enabled.

This mode is used by **rlogin**(1C) and **rlogind**(8C) to implement a remote-echoed, locally ^S/^Q flow-controlled remote login with proper back-flushing of output when interrupts occur; it can be used by other similar programs.

**TIOCREMOTE**
The argument is a pointer to an **int**. If the value of the **int** is non-zero, *remote* mode is enabled; if the value of the **int** is zero, remote mode is disabled. This mode can be enabled or disabled independently of packet mode. When a pseudo-terminal is in remote mode, input to the slave device of the pseudo-terminal is flow controlled and not input edited (regardless of the mode the slave side of the pseudo-terminal). Each write to the controller device produces a record boundary for the process reading the slave device. In normal usage, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an **EOF** character. Note: this means that a process writing to a pseudo-terminal controller in *remote* mode must keep track of line boundaries, and write only one line at a time to the controller. If, for example, it were to buffer up several **NEWLINE** characters and write them to the controller with one **write**( ), it would appear to a process reading from the slave as if a single line containing several **NEWLINE** characters had been typed (as if, for example, a user had typed the **LNEXT** character before typing all but the last of those **NEWLINE** characters). Remote mode can be used when doing remote line editing in a window manager, or whenever flow controlled input is required.

The **ioctl**s **TIOCGWINSZ**, **TIOCSWINSZ**, and, on Sun systems, **TIOCCONS**, can be performed on the controller device of a pseudo-terminal; they have the same effect as when performed on the slave device.

**FILES**

| | |
|---|---|
| **/dev/pty[p-s][0-9a-f]** | pseudo-terminal controller devices |
| **/dev/tty[p-s][0-9a-f]** | pseudo-terminal slave devices |
| **/dev/console** | |

**SEE ALSO**

**rlogin**(1C), **termio**(4), **ldterm**(4M), **ttcompat**(4M), **rlogind**(8C)

**BUGS**

It is apparently not possible to send an **EOT** by writing zero bytes in **TIOCREMOTE** mode.

## NAME

root – pseudo-driver for Sun386i root disk

## CONFIG

**pseudo-device rootdev**

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

The **root** pseudo-driver provides indirect, device-independent access to the root disk on a diskful Sun workstation. The root disk is the disk where the mounted root partition resides - typically the disk from which the system was booted.

The intent of the **root** device is to allow uniform access to the partitions on the root disk, regardless of the disk's controller type or unit number. For example, the following version of **/etc/fstab** will work for any disk (assuming the disk has the standard partitions and filesystems):

```
/dev/roota /      4.2 rw 1 1
/dev/rootg /usr   4.2 ro 1 2
/dev/rooth /export 4.2 rw 1 3
```

When the root device is opened, the open and all subsequent operations on that device (**read**(2V), **write**(2V), **ioctl**(2), **close**(2)) are redirected to the real disk. Therefore, all device-dependent operations on a particular disk are still accessible via the root device (see **dkio**(4S)).

## FILES

| | |
|---|---|
| **/dev/root[a-h]** | block partitions |
| **/dev/rroot[a-h]** | raw partitions |

## SEE ALSO

**fstab**(5), **sd**(4S), **open**(2V), **dkio**(4S),

## NAME

routing – system supporting for local network packet routing

## DESCRIPTION

The network facilities provided general packet routing, leaving routing table maintenance to applications processes.

A simple set of data structures comprise a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. A user process, the routing daemon, maintains this data base with the aid of two socket specific **ioctl**(2) commands, **SIOCADDRT** and **SIOCDELRT**. The commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by super-user.

A routing table entry has the following form, as defined in **<net/route.h>**:

```
struct rtentry {
        u_long  rt_hash;
        struct  sockaddr rt_dst;
        struct  sockaddr rt_gateway;
        short   rt_flags;
        short   rt_refcnt;
        u_long  rt_use;
        struct  ifnet *rt_ifp;
};
```

with *rt_flags* defined from:

```
#define RTF_UP          0x1             /* route usable */
#define RTF_GATEWAY     0x2             /* destination is a gateway */
#define RTF_HOST        0x4             /* host entry (net otherwise) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). When the system is booted, each network interface autoconfigured installs a routing table entry when it wishes to have packets sent through it. Normally the interface specifies the route through it is a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (*rt_refcnt* is non-zero), the resources associated with it will not be reclaimed until all references to it are removed.

The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existent entry, or ENOBUFS if insufficient resources were available to install a new route.

User processes read the routing tables through the **/dev/kmem** device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

## FILES

**/dev/kmem**

## SEE ALSO

**ioctl**(2), **route**(8C), **routed**(8C)

NAME

sd – Disk driver for SCSI Disk Controllers

CONFIG — SUN-3 SYSTEM

**controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40**
**controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40**
**controller si0 at obio ? csr 0x140000 priority 2**
**disk sd0 at sc0 drive 0 flags 0**
**disk sd1 at sc0 drive 1 flags 0**
**disk sd0 at si0 drive 0 flags 0**
**disk sd1 at si0 drive 1 flags 0**
**disk sd2 at sc0 drive 8 flags 0**
**disk sd2 at si0 drive 8 flags 0**

The first two **controller** lines above specify the first SCSI host adapter on a Sun-3/160 system. The third **controller** line above specifies the first and only SCSI host adapter on a Sun-3/50 system. The first four **disk** lines specify the first and second disk drives on the first SCSI controller in a system. The last two **disk** lines specify the first disk drive on the second SCSI controller in a system.

The **drive** value is calculated using the formula:

$8 * target + unit$

where *target* is the SCSI target (controller number on host adapter), and *unit* is the SCSI logical unit.

CONFIG — SUN-2 SYSTEM

**controller sc0 at mbmem ? csr 0x80000 priority 2**
**controller sc1 at mbmem ? csr 0x84000 priority 2**
**controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40**
**disk sd0 at sc0 drive 0 flags 0**
**disk sd1 at sc0 drive 1 flags 0**
**disk sd2 at sc1 drive 0 flags 0**
**disk sd3 at sc1 drive 1 flags 0**

The first two **controller** lines above specify the first and second SCSI host adapters on a Sun-2/120 or Sun-2/170 system. The third **controller** line above specifies the first host adapter on a Sun-2/160 system. The four **disk** lines specify the first and second disk drives on the first and second SCSI controllers in a system (where each SCSI controller is on a different host adapter).

CONFIG — Sun386i

**controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2**
**disk sd0 at wds0 drive 0 flags 0**
**disk sd1 at wds0 drive 8 flags 0**
**disk sd2 at wds0 drive 16 flags 0**

The **drive** value is calculated as described above.

DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0. The standard device names begin with "sd" followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block-files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra '**r**.'

In raw I/O, requests to the SCSI disk must have an offset on a 512 byte boundary, and their length must be a multiple of 512 bytes or the driver will return an error (EINVAL). Likewise **seek** calls should specify a multiple of 512 bytes.

**Disk Support**

On Sun-2, Sun-3, Sun-4 systems, this driver handles all ST-506 and ESDI drives (assuming the correct controller is installed), by reading a label from sector 0 of the drive which describes the disk geometry and partitioning.

On Sun386i systems, this driver supports the CDC Wren III half-height, and Wren IV full-height drives, which have embedded, CCS-compatible SCSI controllers.

The **sd?a** partition is normally used for the root file system on a disk, the sd?b partition as a paging area, and the **sd?c**
partition for pack-pack copying (it normally maps the entire disk). The rest of the disk is normally the **sd?g** partition.

**FILES**

| | |
|---|---|
| **/dev/sd[0-7][a-h]** | block files |
| **/dev/rsd[0-7][a-h]** | raw files |

**SEE ALSO**

**dkio**(4S)

Adaptec ACB 4000 and 5000 Series Disk Controllers OEM Manual (Sun-2, Sun-3, Sun-4 systems only)
Emulex MD21 SCSI Disk Controller Programmer Reference Manual (Sun-2, Sun-3, Sun-4 systems only)
Product Specification for Wren III SCSI Model 94211 (Sun386i systems only)
Product Specification for Wren IV SCSI Model 94171 (Sun386i systems only)

**DIAGNOSTICS**

**sd%d%c:** *cmd how (msg)* **starting blk %d, blk %d (abs blk %d).**

A command such as read or write encountered a error condition (how): either it *failed*, the unit was *restored*, or an operation was *retry*'ed. The *msg* is derived from the error number given by the controller, indicating a condition such as "drive not ready" or "sector not found". The *starting blk* is the first sector of the erroneous command, relative to the beginning of the partition involved. The *blk* is the sector in error, again relative to the beginning of the partition involved. The *abs blk* is the absolute block number of the sector in error.

## NAME

sockio – ioctls that operate directly on sockets

## SYNOPSIS

**#include <sys/sockio.h>**

## DESCRIPTION

The IOCTL's listed in this manual page apply directly to sockets, independent of any underlying protocol. Note: the **setsockopt** system call (see **getsockopt**(2)) is the primary method for operating on sockets as such, rather than on the underlying protocol or network interface. **ioctl**s for a specific network interface or protocol are documented in the manual page for that interface or protocol.

**SIOCSPGRP** The argument is a pointer to an **int**. Set the process-group ID that will subsequently receive **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl** to the value of that **int**.

**SIOCGPGRP** The argument is a pointer to an **int**. Set the value of that **int** to the process-group ID that is receiving **SIGIO** or **SIGURG** signals for the socket referred to by the descriptor passed to **ioctl**.

**SIOCCATMARK** The argument is a pointer to an **int**. Set the value of that **int** to 1 if the read pointer for the socket referred to by the descriptor passed to **ioctl** points to a mark in the data stream for an out-of-band message, and to 0 if it does not point to a mark.

## SEE ALSO

**ioctl**(2), **getsockopt**(2), **filio**(4)

NAME

st – Driver for Sysgen SC 4000 (Archive) and the Emulex MT-02 Tape Controller

CONFIG — SUN-3 SYSTEM

**controller sc0 at vme24d16 ? csr 0x200000 priority 2 vector scintr 0x40**
**controller si0 at vme24d16 ? csr 0x200000 priority 2 vector siintr 0x40**
**controller si0 at obio ? csr 0x140000 priority 2**
**tape st0 at sc0 drive 32 flags 1**
**tape st0 at si0 drive 32 flags 1**
**tape st1 at sc0 drive 40 flags 1**
**tape st1 at si0 drive 40 flags 1**

The first two **controller** lines above specify the first SCSI controller on a Sun-3/160 system. The third **controller** line above specifies the first and only SCSI controller on a Sun-3/50 system. The four **tape** lines specify the first and second tape drives on the first SCSI controller in a system.

The **drive** value is calculated using the formula:

$8 * target + unit$

where *target* is the SCSI target, and *unit* is the SCSI logical unit.

CONFIG — SUN-2 SYSTEM

**controller sc0 at mbmem ? csr 0x80000 priority 2**
**controller sc0 at vme24 ? csr 0x200000 priority 2 vector scintr 0x40**
**controller sc1 at mbmem ? csr 0x84000 priority 2**
**tape st0 at sc0 drive 32 flags 1**
**tape st0 at sc1 drive 32 flags 1**
**tape st1 at sc0 drive 40 flags 1**
**tape st1 at sc1 drive 40 flags 1**

The first two **controller** lines above specify the first and second SCSI controllers on a Sun-2/120 or Sun-2/170 system. The third **controller** line specifies the first controller on a Sun-2/160 system. The four **tape** lines specify the first and second tape drives on the first and second SCSI controllers in a system.

The **drive** value is calculated as described above.

CONFIG — Sun386i

**controller wds0 at obmem ? csr 0xFB000000 dmachan 7 irq 16 priority 2**
**tape st0 at wds0 drive 32 flags 1**

The **drive** value is calculated as described above.

DESCRIPTION

The Sysgen tape controller is a SCSI bus interface to an Archive streaming tape drive. It provides a standard tape interface to the device, see **mtio**(4), with some deficiencies listed under BUGS below. To utilize the QIC 24 format, access the logical device that is eight more than the default physical (QIC 11) device (that is, rst0 = QIC 11, rst8 = QIC 24). QIC 24 is the preferred format on Sun386i systems.

FILES

| | |
|---|---|
| **/dev/rst[0-3]** | QIC 11 Format |
| **/dev/rst[8-11]** | QIC 24 Format |
| **/dev/nrst[0-3]** | non-rewinding QIC 11 Format |
| **/dev/nrst[8-11]** | non-rewinding QIC 24 Format |

SEE ALSO

**mtio**(4)

Sysgen SC4000 Intelligent Tape Controller Product Specification

DIAGNOSTICS

**st*: tape not online.**
**st*: no cartridge loaded.**

**st*: cartridge is write protected.**
**st*: format change failed.**
**st*: device not supported.**

BUGS

The tape cannot reverse direction so the **BSF** and **BSR** ioctls are not supported.

The **FSR** ioctl is not supported.

Most disk I/O over the SCSI bus is prevented when the tape is in use. This is because the controller does not free the bus while the tape is in motion (even during rewind).

When using the raw device, the number of bytes in any given transfer must be a multiple of 512. If it is not, the device driver returns an error.

The driver will only write an end of file mark on close if the last operation was a write, without regard for the mode used when opening the file. Empty files will be deleted on a raw tape copy operation.

Some older systems may not support the QIC 24 device, and may complain (or exhibit erratic behavior) when the user attempts a QIC 24 device access.

**NAME**

streamio – STREAMS ioctl commands

**SYNOPSIS**

**#include <stropts.h>**
**int ioctl (fildes, command, arg)**
**int fildes, command;**

**DESCRIPTION**

STREAMS (see **intro**(2)) ioctl commands are a subset of **ioctl**(2) commands that perform a variety of control functions on STREAMS. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream*head. Certain combinations of these arguments may be passed to a module or driver in the stream.

*fildes* is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the stream referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. Subsequent system calls will fail with *errno* set to this value.

**IOCTLS**

The following **ioctl** commands, with error values indicated, are applicable to all STREAMS files:

**I_PUSH** Pushes the module whose name is pointed to by *arg* onto the top of the current stream, just below the *stream*head. It then calls the open routine of the newly-pushed module.

**I_PUSH** will fail if one of the following occurs:

| | |
|---|---|
| EINVAL | The module name is invalid. |
| EFAULT | *arg* points outside the allocated address space. |
| ENXIO | The open routine of the new module failed. |
| ENXIO | A hangup is received on the stream referred to by *fildes*. |

**I_POP** Removes the module just below the *stream head* of the stream pointed to by *fildes*. *arg* should be 0 in an **I_POP** request.

**I_POP** will fail if one of the following occurs:

| | |
|---|---|
| EINVAL | No module is present on *stream*. |
| ENXIO | A hangup is received on the stream referred to by *fildes*. |

**I_LOOK** Retrieves the name of the module just below the *stream head* of the stream pointed to by *fildes*, and places it in a NULL terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least **FMNAMESZ+1** bytes long. An '**#include <sys/conf.h>**' declaration is required.

**I_LOOK** will fail if one of the following occurs:

| | |
|---|---|
| EFAULT | *arg* points outside the allocated address space of the process. |
| EINVAL | No module is present on *stream*. |

**I_FLUSH** This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

| | |
|---|---|
| **FLUSHR** | Flush read queues. |
| **FLUSHW** | Flush write queues. |
| **FLUSHRW** | Flush read and write queues. |

**I_FLUSH** will fail if one of the following occurs:

| | |
|---|---|
| EAGAIN | No buffers could be allocated for the flush message. |
| EINVAL | The value of *arg* is invalid. |
| ENXIO | A hangup is received on the stream referred to by *fildes*. |

**I_SETSIG** Informs the *stream head* that the user wishes the kernel to issue the **SIGPOLL** signal (see **sigvec**(2)) when a particular event has occurred on the stream associated with *fildes*. **I_SETSIG** supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

| | |
|---|---|
| **S_INPUT** | A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length. |
| **S_HIPRI** | A priority message is present on the *stream head* read queue. This is set even if the message is of zero length. |
| **S_OUTPUT** | The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream. |
| **S_MSG** | A STREAMS signal message that contains the **SIGPOLL** signal has reached the front of the *stream head* read queue. |

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value **S_HIPRI**.

Processes that wish to receive **SIGPOLL** signals must explicitly register to receive them using **I_SETSIG**. If several processes register to receive this signal for the same event on the same *stream*, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further **SIGPOLL** signals.

**I_SETSIG** will fail if one of the following occurs:

| | |
|---|---|
| EINVAL | The value of *arg* is invalid or *arg* is zero and the process is not registered to receive the **SIGPOLL** signal. |
| EAGAIN | A data structure could not be allocated to store the signal request. |

**I_GETSIG** Returns the events for which the calling process is currently registered to be sent a **SIGPOLL** signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of **I_SETSIG** above.

**I_GETSIG** will fail if one of the following occurs:

EINVAL — The process is not registered to receive the **SIGPOLL** signal.

EFAULT — *arg* points outside the allocated address space of the process.

**I_FIND** This request compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present.

**I_FIND** will fail if one of the following occurs:

EFAULT — *arg* points outside the allocated address space of the process.

EINVAL — *arg* does not point to a valid module name.

**I_PEEK** This request allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures (see **getmsg**(2)) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to **RS_HIPRI, I_PEEK** will only look for a priority message on the *stream head* read queue.

**I_PEEK** returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the **RS_HIPRI** flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or **RS_HIPRI.**

**I_PEEK** will fail if one of the following occurs:

EFAULT — *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

**I_SRDOPT** Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

**RNORM** — Byte-stream mode, the default.

**RMSGD** — Message-discard mode.

**RMSGN** — Message-nondiscard mode.

Read modes are described in **read**(2V).

**I_SRDOPT** will fail if one of the following occurs:

EINVAL — *arg* is not one of the above legal values.

**I_GRDOPT** Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in **read**(2V).

**I_GRDOPT** will fail if one of the following occurs:

EFAULT — *arg* points outside the allocated address space of the process.

**I_NREAD** Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the **ioctl** return value is greater than zero, this indicates that a zero-length message is next on the queue.

**I_NREAD** will fail if one of the following occurs:

EFAULT *arg* points outside the allocated address space of the process.

**I_FDINSERT** creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

*arg* points to a *strfdinsert* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
int             fd;
int             offset;
```

The *len* field in the *ctlbuf strbuf* structure (see **putmsg**(2)) must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fd* specifies the file descriptor of the other stream and *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where **I_FDINSERT** will store a pointer to the *fd* stream's driver read queue structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

*flags* specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to **RS_HIPRI**. For non-priority messages, **I_FDINSERT** will block if the stream write queue is full due to internal flow control conditions. For priority messages, **I_FDINSERT** does not block on this condition. For non-priority messages, **I_FDINSERT** does not block when the write queue is full and **O_NDELAY** is set. Instead, it fails and sets *errno* to EAGAIN.

**I_FDINSERT** also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether **O_NDELAY** has been specified. No partial message is sent.

**I_FDINSERT** will fail if one of the following occurs:

EAGAIN A non-priority message was specified, the **O_NDELAY** flag is set, and the stream write queue is full due to internal flow control conditions.

EAGAIN Buffers could not be allocated for the message that was to be created.

EFAULT *arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space of the process.

EINVAL *fd* in the *strfdinsert* structure is not a valid, open stream file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*;

*offset* does not specify a properly-aligned location in the data buffer; an undefined value is pointed to by *flags*.

ENXIO — A hangup is received on the stream referred to by *fildes*.

ERANGE — The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

**I_STR** Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to permit a process to specify timeouts and variable-sized amounts of data when sending an **ioctl** request to downstream modules and drivers. It allows information to be sent with the *ioctl*, and will return to the user any information sent upstream by the downstream recipient. **I_STR** blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one **I_STR** can be active on a stream. Further **I_STR** calls will block until the active **I_STR** completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The **O_NDELAY** (see **open**(2V)) flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;     /* downstream command */
int     ic_timout;  /* ACK/NAK timeout */
int     ic_len;     /* length of data arg */
char    *ic_dp;     /* ptr to data arg */
```

*ic_cmd* is the internal ioctl command intended for a downstream module or driver and *ic_timout* is the number of seconds (–1 = infinite, 0 = use default, >0 = as specified) an **I_STR** request will wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream.

**I_STR** will fail if one of the following occurs:

EAGAIN — Buffers could not be allocated for the **ioctl** message.

EFAULT — *arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space of the process.

EINVAL — *ic_len* is less than 0 or *ic_len* is larger than the maximum

configured size of the data part of a message or *ic_timout* is less than −1.

ENXIO — A hangup is received on the stream referred to by *fildes*.

ETIME — A downstream **ioctl** timed out before acknowledgement was received.

An **I_STR** can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *stream*head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the **ioctl** command sent downstream fails. For these cases, **I_STR** will fail with *errno* set to the value in the message.

**I_SENDFD** Requests the stream associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

**I_SENDFD** converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see **intro**(2)) of the *stream head* at the other end of the stream pipe to which it is connected.

**I_SENDFD** will fail if one of the following occurs:

EAGAIN — The sending stream is unable to allocate a message block to contain the file pointer.

EAGAIN — The read queue of the receiving *stream head* is full and cannot accept the message sent by **I_SENDFD**.

EBADF — *arg* is not a valid, open file descriptor.

EINVAL — *fildes* is not connected to a stream pipe.

ENXIO — A hangup is received on the stream referred to by *fildes*.

**I_RECVFD** Retrieves the file descriptor associated with the message sent by an **I_SENDFD** **ioctl** over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

*fd* is an integer file descriptor. *uid* and *gid* are the user ID and group ID, respectively, of the sending stream.

If **O_NDELAY** is not set (see **open**(2V)), **I_RECVFD** will block until a message is present at the *stream*head. If **O_NDELAY** is set, **I_RECVFD** will fail with *errno* set to EAGAIN if no message is present at the *stream*head.

If the message at the *stream head* is a message sent by an **I_SENDFD**, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*.

**I_RECVFD** will fail if one of the following occurs:

EAGAIN — A message was not present at the *stream head* read queue, and the **O_NDELAY** flag is set.

| | |
|---|---|
| EBADMSG | The message at the *stream head* read queue was not a message containing a passed file descriptor. |
| EFAULT | *arg* points outside the allocated address space of the process. |
| EMFILE | Too many descriptors are active. |
| ENXIO | A hangup is received on the stream referred to by *fildes*. |

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

**I_LINK** Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. **I_LINK** causes the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see **I_UNLINK**) on success, and a −1 on failure.

**I_LINK** will fail if one of the following occurs:

| | |
|---|---|
| ENXIO | A hangup is received on the stream referred to by *fildes*. |
| ETIME | The **ioctl** timed out before an acknowledgement was received. |
| EAGAIN | Storage could not be allocated to perform the **I_LINK**. |
| EBADF | *arg* is not a valid, open file descriptor. |
| EINVAL | The stream referred to by *fildes* does not support multiplexing. |
| EINVAL | *arg* is not a stream, or is already linked under a multiplexor. |
| EINVAL | The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place. |

An **I_LINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_LINK** will fail with *errno* set to the value in the message.

**I_UNLINK** Disconnects the two streams specified by *fildes* and *arg*. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the **ioctl** **I_LINK** command when a stream was linked below the multiplexing driver. If *arg* is −1, then all streams which were linked to *fildes* are disconnected. As in **I_LINK**, this command requires the multiplexing driver to acknowledge the unlink.

**I_UNLINK** will fail if one of the following occurs:

| | |
|---|---|
| ENXIO | A hangup is received on the stream referred to by *fildes*. |
| ETIME | The **ioctl** timed out before an acknowledgement was received. |
| EAGAIN | Buffers could not be allocated for the acknowledgement message. |

EINVAL The multiplexor ID number was invalid.

An **I_UNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_UNLINK** will fail with *errno* set to the value in the message.

SEE ALSO

**close**(2), **fcntl**(2V), **getmsg**(2), **intro**(2), **ioctl**(2), **open**(2V), **poll**(2), **putmsg**(2), **read**(2V), **sigvec**(2), **write**(2V)

*STREAMS Programmer's Guide*
*STREAMS Primer*

**NAME**

taac – Sun applications accelerator

**CONFIG**

**taac0 at vme32d32 ? csr 0x28000000**

**DESCRIPTION**

The **taac** interface supports the optional TAAC-1 Applications Accelerator. This add-on device is composed of a very-long-instruction-word computation engine, coupled with an 8MB memory array. This memory area can be used either as a frame buffer, or as storage for large data sets.

Programs can be downloaded for execution on the TAAC-1 directly, they can be executed by the host processor, or the host processor and the TAAC-1 engine can be used in combination. See the *TAAC-1 User's Guide* for detailed information on accessing the TAAC-1 from the host. This manual also describes the C compiler, the programming tools, and the support libraries for the TAAC-1.

Programs on the host processor gain access to the TAAC-1 registers and memory by using **mmap**(2).

**FILES**

**/dev/taac0**
**/usr/include/taac1**
**/usr/lib/taac1**

**SEE ALSO**

**mmap**(2)

*TAAC-1 Application Accelerator: User Guide*

## NAME

tcp – Internet Transmission Control Protocol

## SYNOPSIS

**#include <sys/socket.h>**
**#include <netinet/in.h>**

**s = socket(AF_INET, SOCK_STREAM, 0);**

## DESCRIPTION

TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol used to support the **SOCK_STREAM** abstraction. TCP is layered above the Internet Protocol (IP), the Internet protocol family's unreliable internetwork datagram delivery protocol.

TCP uses IP's host-level addressing and adds its own per-host collection of "port addresses". The endpoints of a TCP connection are identified by the combination of an IP address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See **inet**(4F) for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either "active" or "passive". Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP address and TCP port number bound with the **bind**(2) system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the **listen**(2) system call after binding the socket with **bind** . This establishes a queueing parameter for the passive socket. After this, connections to the passive socket can be received with the **accept**(2) system call. Active sockets use the **connect**(2) call after binding to initiate connections.

By using the special value **INADDR_ANY**, the local IP address can be left unspecified in the **bind** call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP address will be bound at connection time to the address of the network interface used to service the connection.

Once a connection has been established, data can be exchanged using the **read**(2V) and **write**(2V) system calls.

TCP supports one socket option which is set with **setsockopt** and tested with **getsockopt**(2). Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, TCP_NODELAY (defined in **<netinet/tcp.h>**), to defeat this algorithm. The option level for the **setsockopt** call is the protocol number for TCP, available from **getprotobyname** (see **getprotoent**(3N)).

Options at the IP level may be used with TCP; see **ip**(4P).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of **send**(2). The caller may mark one byte as "urgent" with the **MSG_OOB** flag to **send**(2). This causes an "urgent pointer" pointing to this byte to be set in the TCP stream. The receiver on the other side of the stream is notified of the urgent data by a SIGURG signal. The **SIOCATMARK** ioctl returns a value indicating whether the stream is at the urgent mark. Because the system never returns data across the urgent mark in a single **read**(2V) call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the **SIOCATMARK** ioctl, until it reaches the mark.

Incoming connection requests that include an IP source route option are noted, and the reverse source route is used in responding.

TCP assumes the datagram service it is layered above is unreliable. A checksum over all data helps TCP implement reliability. Using a window-based flow control mechanism that makes use of positive acknowledgements, sequence numbers, and a retransmission strategy, TCP can usually recover when datagrams are damaged, delayed, duplicated or delivered out of order by the underlying communication medium.

If the local TCP receives no acknowledgements from its peer for a period of time, as would be the case if the remote machine crashed, the connection is closed and an error is returned to the user. If the remote machine reboots or otherwise loses state information about a TCP connection, the connection is aborted and an error is returned to the user.

## ERRORS

A socket operation may fail if:

| | |
|---|---|
| EISCONN | A **connect** operation was attempted on a socket on which a **connect** operation had already been performed. |
| ETIMEDOUT | A connection was dropped due to excessive retransmissions. |
| ECONNRESET | The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash). |
| ECONNREFUSED | The remote peer actively refused connection establishment (usually because no process is listening to the port). |
| EADDRINUSE | A **bind** operation was attempted on a socket with a network address/port pair that has already been bound to another socket. |
| EADDRNOTAVAIL | A **bind** operation was attempted on a socket with a network address for which no network interface exists. |
| EACCES | A **bind** operation was attempted with a "reserved" port number and the effective user ID of the process was not super-user. |
| ENOBUFS | The system ran out of memory for internal data structures. |

## SEE ALSO

**accept**(2), **bind**(2), **connect**(2), **getsockopt**(2), **listen**(2), **read**(2V), **send**(2), **write**(2V), **getprotoent**(3N), **inet**(4F), **ip**(4P)

Postel, Jon, *Transmission Control Protocol - DARPA Internet Program Protocol Specification*, RFC 793, Network Information Center, SRI International, Menlo Park, Calif., September 1981.

## BUGS

**SIOCSHIWAT** and **SIOCGHIWAT** ioctl's to set and get the high water mark for the socket queue, and so that it can be changed from 2048 bytes to be larger or smaller, have been defined (in **<sys/ioctl.h>**) but not implemented.

## NAME

termio – general terminal interface

## SYNOPSIS

**#include <sys/termios.h>**

## DESCRIPTION

Asynchronous communications ports, pseudo-terminals, and the special interface accessed by **/dev/tty** all use the same general interface, no matter what hardware (if any) is involved. The remainder of this section discusses the common features of this interface.

### Opening a Terminal Device File

When a terminal file is opened, the process normally waits until a connection is established. (In practice, users' programs seldom open these files; they are opened by **getty**(8) and become a user's standard input, output, and error files.) If the **O_NDELAY** flag was set in the second argument to **open**(2V), the **open**() will complete immediately without waiting for a connection to be established.

### Process Groups

A terminal may have a distinguished process group associated with it. This distinguished process group plays a special role in handling signal-generating input characters, as discussed below in the **Special Characters** section below.

A command interpreter, such as **csh**(1), that supports "job control" can allocate the terminal to different *jobs*, or process groups, by placing related processes in a single process group and associating this process group with the terminal. A terminal's associated process group may be set or examined by a process with sufficient privileges. The terminal interface aids in this allocation by restricting access to the terminal by processes that are not in the current process group; see **Job Access Control** below.

### The Controlling Terminal

A terminal may belong to a process as its *controlling terminal*. If a process that is a "session process group leader", and that does not have a controlling terminal, opens a terminal file not already associated with a process group, the terminal associated with that terminal file becomes the controlling terminal for that process, and the terminal's distinguished process group is set to the process group of that process. (Currently, this also happens if a process that does not have a controlling terminal and is not a member of a process group opens a terminal. In this case, if the terminal is not associated with a process group, a new process group is created with a process group ID equal to the process ID of the process in question, and the terminal is assigned to that process group. The process is made a member of the terminal's process group.)

The controlling terminal is inherited by a child process during a **fork**(2). A process relinquishes its control terminal when it changes its process group using **setpgrp**(2V) or when it issues a **TIOCNOTTY** **ioctl**(2) call on a file descriptor created by opening the file **/dev/tty**.

When a session process group leader that has a controlling terminal terminates, the distinguished process group of the controlling terminal is set to zero (indicating no distinguished process group). This allows the terminal to be acquired as a controlling terminal by a new session process group leader.

### Closing a Terminal Device File

When a terminal device file is closed, the process closing the file waits until all output is drained; all pending input is then flushed, and finally a disconnect is performed. If **HUPCL** is set, the existing connection is severed (by hanging up the phone line, if appropriate).

### Job Access Control

If a process is in the (non-zero) distinguished process group of its controlling terminal, or if the terminal's distinguished process group is zero (if either of these are true, the process is said to be a *foreground process*), then **read**(2V) operations are allowed as described below in **Input Processing and Reading Characters**. If a process is not in the (non-zero) distinguished process group of its controlling terminal (if this is true, the process is said to be a *background process*), then any attempts to read from that terminal will send that process' process group a **SIGTTIN** signal, unless the process is ignoring **SIGTTIN**, has **SIGTTIN** blocked, or is in the middle of process creation using **vfork**(2); in that case, the read will return −1 and set **errno** to **EIO**, and the **SIGTTIN** signal will not be sent. The **SIGTTIN** signal will normally stop the

members of that process group.

When the **TOSTOP** bit is set in the **c_lflag** field, attempts by a background process to write to its controlling terminal will send that process' process group a **SIGTTOU** signal, unless the process is ignoring **SIGTTOU**, has **SIGTTOU** blocked, or is in the middle of process creation using **vfork()**; in that case, the process will be allowed to write to the terminal and the **SIGTTOU** signal will not be sent. The **SIGTTOU** signal will normally stop the members of that process group. Certain **ioctl()** calls that set terminal parameters are treated in this same fashion, except that **TOSTOP** is not checked; the effect is identical to that of terminal writes when **TOSTOP** is set. See **IOCTLS**.

**Input Processing and Reading Characters**

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. If the **IMAXBEL** mode has not been selected, all the saved characters are thrown away without notice when the input limit is reached; if the **IMAXBEL** mode has been selected, the driver refuses to accept any further input, and echoes a bell (ASCII BEL).

Two general kinds of input processing are available, determined by whether the terminal device file is in canonical mode or non-canonical mode (see **ICANON** in the **Local Modes** section).

The style of input processing can also be very different when the terminal is put in non-blocking I/O mode; see **read**(2V). In this case, reads from the terminal will never block.

It is possible to simulate terminal input using the **TIOCSTI ioctl()** call, which takes, as its third argument, the address of a character. The system pretends that this character was typed on the argument terminal, which must be the process' controlling terminal unless the process' effective user ID is super-user.

**Canonical Mode Input Processing**

In canonical mode input processing, terminal input is processed in units of lines. A line is delimited by a NEWLINE (ASCII LF) character, an EOF (by default, an ASCII EOT) character, or one of two user-specified end-of-line characters, **EOL** and **EOL2**. This means that a **read()** will not complete until an entire line has been typed or a signal has been received. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

Erase and kill processing occurs during input. The **ERASE** character (by default, the character DEL) erases the last character typed in the current input line. The **WERASE** character (by default, the character CTRL-W) erases the last "word" typed in the current input line (but not any preceding SPACE or TAB characters). A "word" is defined as a sequence of non-blank characters, with TAB characters counted as blanks. Neither **ERASE** nor **WERASE** will erase beyond the beginning of the line. The **KILL** character (by default, the character CTRL-U) kills (deletes) the entire current input line, and optionally outputs a NEWLINE character. All these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done.

The **REPRINT** character (the character CTRL-R) prints a NEWLINE followed by all characters that have not been read. Reprinting also occurs automatically if characters that would normally be erased from the screen are fouled by program output. The characters are reprinted as if they were being echoed; as a consequence, if **ECHO** is not set, they are not printed.

The **ERASE** and **KILL** characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The **ERASE** and **KILL** characters may be changed.

**Non-Canonical Mode Input Processing**

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The **MIN** and **TIME** values are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the read is satisfied (when the characters are returned to the user). TIME is a timer of 0.10 second granularity that is used to timeout bursty and short term data transmissions. The four possible values for MIN and TIME and their interactions are described below.

**Case A: MIN > 0, TIME > 0**

In this case TIME serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between MIN and TIME is as follows: as soon as one character is received, the intercharacter timer is started. If MIN characters are received before the intercharacter timer expires (remember that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before MIN characters are received, the characters received to that point are returned to the user. Note: if MIN expires at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (MIN > 0, TIME > 0) the read will sleep until the MIN and TIME mechanisms are activated by the receipt of the first character.

**Case B: MIN > 0, TIME = 0**

In this case, since the value of TIME is zero, the timer plays no role and only MIN is significant. A pending read is not satisfied until MIN characters are received (the pending read will sleep until MIN characters are received). A program that uses this case to read record-based terminal I/O may block indefinitely in the read operation.

**Case C: MIN = 0, TIME > 0**

In this case, since MIN = 0, TIME no longer represents an intercharacter timer. It now serves as a read timer that is activated as soon as a **read()** is done. A read is satisfied as soon as a single character is received or the read timer expires. Note: in this case if the timer expires, no character will be returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case the read will not block indefinitely waiting for a character – if no character is received within TIME*.10 seconds after the read is initiated, the read will return with zero characters.

**Case D: MIN = 0, TIME = 0**

In this case return is immediate. The minimum of either the number of characters requested or the number of characters currently available will be returned without waiting for more characters to be input.

**Comparison of the Different Cases of MIN, TIME Interaction**

Some points to note about MIN and TIME:

1. In the following explanations one may notice that the interactions of MIN and TIME are not symmetric. For example, when MIN > 0 and TIME = 0, TIME has no effect. However, in the opposite case where MIN = 0 and TIME > 0, both MIN and TIME play a role in that MIN is satisfied with the receipt of a single character.
2. Also note that in case A (MIN > 0, TIME > 0), TIME represents an intercharacter timer while in case C (TIME = 0, TIME > 0) TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, where MIN > 0, exist to handle burst mode activity (for example, file transfer programs) where a program would like to process at least MIN characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; while in case B, it is turned off.

Cases C and D exist to handle single character timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C the read is timed; while in case D, it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length. That is, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20 characters will be returned to the user.

**Writing Characters**

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed as they are typed if echoing has been enabled. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

**Special Characters**

Certain characters have special functions on input and/or output. These functions and their default character values are summarized as follows:

**INTR** (CTRL-C or ASCII ETX) generates a **SIGINT** signal, which is sent to all processes in the distinguished process group associated with the terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see **sigvec**(2).

**QUIT** (CTRL-| or ASCII FS) generates a **SIGQUIT** signal, which is sent to all processes in the distinguished process group associated with the terminal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

**ERASE** (Rubout or ASCII DEL) erases the preceding character. It will not erase beyond the start of a line, as delimited by a **NL**, **EOF**, **EOL**, or **EOL2** character.

**WERASE** (CTRL-W or ASCII ETB) erases the preceding "word". It will not erase beyond the start of a line, as delimited by a **NL**, **EOF**, **EOL**, or **EOL2** character.

**KILL** (CTRL-U or ASCII NAK) deletes the entire line, as delimited by a **NL**, **EOF**, **EOL**, or **EOL2** character.

**REPRINT** (CTRL-R or ASCII DC2) reprints all characters that have not been read, preceded by a NEWLINE.

**EOF** (CTRL-D or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a NEWLINE, and the **EOF** is discarded. Thus, if there are no characters waiting, which is to say the **EOF** occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

**NL** (ASCII LF) is the normal line delimiter. It can not be changed; it can, however, be escaped by the **LNEXT** character.

**EOL**
**EOL2** (ASCII NUL) are additional line delimiters, like **NL**. They are not normally used.

**SUSP** (CTRL-Z or ASCII EM) is used by the job control facility to change the current job to return to the controlling job. It generates a **SIGTSTP** signal, which stops all processes in the terminal's process group.

**STOP** (CTRL-S or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, **STOP** characters are ignored and not read.

**START** (CTRL-Q or ASCII DC1) is used to resume output that has been suspended by a **STOP** character. While output is not suspended, **START** characters are ignored and not read.

**DISCARD** (CTRL-O or ASCII SI) causes subsequent output to be discarded until another **DISCARD** character is typed, more input arrives, or the condition is cleared by a program.

**LNEXT** (CTRL-V or ASCII SYN) causes the special meaning of the next character to be ignored; this works for all the special characters mentioned above. This allows characters to be input that would otherwise get interpreted by the system (for example, **KILL**, **QUIT**.)

The character values for **INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SUSP, STOP, START, DISCARD,** and **LNEXT** may be changed to suit individual tastes. If the value of a special control character is 0, the function of that special control character will be disabled. The **ERASE, KILL,** and **EOF** characters may be escaped by a preceding \ character, in which case no special function is done. Any of the special characters may be preceded by the **LNEXT** character, in which case no special function is done.

**Modem Disconnect**

If a modem disconnect is detected, and the CLOCAL flag is not set in the **c_cflag** field, a **SIGHUP** signal is sent to all processes in the distinguished process group associated with this terminal. Unless other arrangements have been made, this signal terminates the processes. If **SIGHUP** is ignored or caught, any subsequent **read()** returns with an end-of-file indication until the terminal is closed. Thus, programs that read a terminal and test for end-of-file can terminate appropriately after a disconnect. Any subsequent **write()** will return −1 and set **errno** to **EIO** until the terminal is closed.

**Terminal Parameters**

The parameters that control the behavior of devices and modules providing the **termios** interface are specified by the **termios** structure, defined by **<sys/termios.h>**. Several **ioctl()** system calls that fetch or change these parameters use this structure:

```
#define  NCCS      17
struct   termios {
         unsigned  long   c_iflag;      /* input modes */
         unsigned  long   c_oflag;      /* output modes */
         unsigned  long   c_cflag;      /* control modes */
         unsigned  long   c_lflag;      /* local modes */
         unsigned  char   c_line;       /* line discipline */
         unsigned  char   c_cc[NCCS];   /* control chars */
};
```

The special control characters are defined by the array **c_cc**. The relative positions and initial values for each function are as follows:

| | | |
|---|---|---|
| 0 | VINTR | ETX |
| 1 | VQUIT | FS |
| 2 | VERASE | DEL |
| 3 | VKILL | NAK |
| 4 | VEOF | EOT |
| 5 | VEOL | NUL |
| 6 | VEOL2 | NUL |
| 7 | VSWTCH | NUL |
| 8 | VSTART | DC1 |
| 9 | VSTOP | DC3 |
| 10 | VSUSP | EM |
| 12 | VREPRINT | DC2 |
| 13 | VDISCARD | SI |
| 14 | VWERASE | ETB |
| 15 | VLNEXT | SYN |

The **MIN** value is stored in the **VMIN** element of the **c_cc** array, and the **TIME** value is stored in the **VTIME** element of the **c_cc** array. The **VMIN** element is the same element as the **VEOF** element, and the **VTIME** element is the same element as the **VEOL** element.

**Input Modes**

The **c_iflag** field describes the basic terminal input control:

| | | |
|---|---|---|
| **IGNBRK** | 0000001 | Ignore break condition. |
| **BRKINT** | 0000002 | Signal interrupt on break. |
| **IGNPAR** | 0000004 | Ignore characters with parity errors. |

| | | |
|---|---|---|
| **PARMRK** | 0000010 | Mark parity errors. |
| **INPCK** | 0000020 | Enable input parity check. |
| **ISTRIP** | 0000040 | Strip character. |
| **INLCR** | 0000100 | Map NL to CR on input. |
| **IGNCR** | 0000200 | Ignore CR. |
| **ICRNL** | 0000400 | Map CR to NL on input. |
| **IUCLC** | 0001000 | Map upper-case to lower-case on input. |
| **IXON** | 0002000 | Enable start/stop output control. |
| **IXANY** | 0004000 | Enable any character to restart output. |
| **IXOFF** | 0010000 | Enable start/stop input control. |
| **IMAXBEL** | 0020000 | Echo BEL on input line too long. |

If **IGNBRK** is set, a break condition (a character framing error with data all zeros) detected on input is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise, if **BRKINT** is set, a break condition will generate a **SIGINT** and flush both the input and output queues. If neither **IGNBRK** nor **BRKINT** is set, a break condition is read as a single ASCII NUL character ('\0').

If **IGNPAR** is set, characters with framing or parity errors (other than break) are ignored. Otherwise, if **PARMRK** is set, a character with a framing or parity error that is not ignored is read as the three-character sequence: '\377', '\0', *X*, where *X* is the data of the character received in error. To avoid ambiguity in this case, if **ISTRIP** is not set, a valid character of '\377' is read as '\377', '\377'. If neither **IGNPAR** nor **PARMRK** is set, a framing or parity error (other than break) is read as a single ASCII NUL character ('\0').

If **INPCK** is set, input parity checking is enabled. If **INPCK** is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If **ISTRIP** is set, valid input characters are first stripped to 7 bits, otherwise all 8 bits are processed.

If **INLCR** is set, a received NL character is translated into a CR character. If **IGNCR** is set, a received CR character is ignored (not read). Otherwise if **ICRNL** is set, a received CR character is translated into a NL character.

If **IUCLC** is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If **IXON** is set, start/stop output control is enabled. A received **STOP** character will suspend output and a received **START** character will restart output. The **STOP** and **START** characters will not be read, but will merely perform flow control functions. If **IXANY** is set, any input character will restart output that has been suspended.

If **IXOFF** is set, the system will transmit a **STOP** character when the input queue is nearly full, and a **START** character when enough input has been read that the input queue is nearly empty again.

If **IMAXBEL** is set, the ASCII BEL character is echoed if the input stream overflows. Further input will not be stored, but any input already present in the input stream will not be disturbed. If **IMAXBEL** is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

The initial input control value is **BRKINT, ICRNL, IXON, ISTRIP.**

**Output modes**

The **c_oflag** field specifies the system treatment of output:

| | | |
|---|---|---|
| **OPOST** | 0000001 | Postprocess output. |
| **OLCUC** | 0000002 | Map lower case to upper on output. |
| **ONLCR** | 0000004 | Map NL to CR-NL on output. |
| **OCRNL** | 0000010 | Map CR to NL on output. |
| **ONOCR** | 0000020 | No CR output at column 0. |
| **ONLRET** | 0000040 | NL performs CR function. |
| **OFILL** | 0000100 | Use fill characters for delay. |
| **OFDEL** | 0000200 | Fill is DEL, else NUL. |

| | | |
|---|---|---|
| **NLDLY** | 0000400 | Select new-line delays: |
| **NL0** | 0 | |
| **NL1** | 0000400 | |
| **CRDLY** | 0003000 | Select carriage-return delays: |
| **CR0** | 0 | |
| **CR1** | 0001000 | |
| **CR2** | 0002000 | |
| **CR3** | 0003000 | |
| **TABDLY** | 0014000 | Select horizontal-tab delays: |
| **TAB0** | 0 | or tab expansion: |
| **TAB1** | 0004000 | |
| **TAB2** | 0010000 | |
| **XTABS** | 0014000 | Expand tabs to spaces. |
| **BSDLY** | 0020000 | Select backspace delays: |
| **BS0** | 0 | |
| **BS1** | 0020000 | |
| **VTDLY** | 0040000 | Select vertical-tab delays: |
| **VT0** | 0 | |
| **VT1** | 0040000 | |
| **FFDLY** | 0100000 | Select form-feed delays: |
| **FF0** | 0 | |
| **FF1** | 0100000 | |

If **OPOST** is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If **OLCUC** is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with **IUCLC**.

If **ONLCR** is set, the NL character is transmitted as the CR-NL character pair. If **OCRNL** is set, the CR character is transmitted as the NL character. If **ONOCR** is set, no CR character is transmitted when at column 0 (first position). If **ONLRET** is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If **OFILL** is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If **OFDEL** is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If **ONLRET** is set, the RETURN delays are used instead of the NEWLINE delays. If **OFILL** is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If **OFILL** is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3, specified by **TAB3** or **XTABS**, specifies that TAB characters are to be expanded into SPACE characters. If **OFILL** is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If **OFILL** is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is **OPOST, ONLCR, XTABS.**

The **c_cflag** field describes the hardware control of the terminal:

| | | |
|---|---|---|
| **CBAUD** | 0000017 | Baud rate: |
| **B0** | 0 | Hang up |
| **B50** | 0000001 | 50 baud |
| **B75** | 0000002 | 75 baud |
| **B110** | 0000003 | 110 baud |
| **B134** | 0000004 | 134.5 baud |
| **B150** | 0000005 | 150 baud |
| **B200** | 0000006 | 200 baud |
| **B300** | 0000007 | 300 baud |
| **B600** | 0000010 | 600 baud |
| **B1200** | 0000011 | 1200 baud |
| **B1800** | 0000012 | 1800 baud |
| **B2400** | 0000013 | 2400 baud |
| **B4800** | 0000014 | 4800 baud |
| **B9600** | 0000015 | 9600 baud |
| **B19200** | 0000016 | 19200 baud |
| **B38400** | 0000017 | 38400 baud |
| **CSIZE** | 0000060 | Character size: |
| **CS5** | 0 | 5 bits |
| **CS6** | 0000020 | 6 bits |
| **CS7** | 0000040 | 7 bits |
| **CS8** | 0000060 | 8 bits |
| **CSTOPB** | 0000100 | Send two stop bits, else one. |
| **CREAD** | 0000200 | Enable receiver. |
| **PARENB** | 0000400 | Parity enable. |
| **PARODD** | 0001000 | Odd parity, else even. |
| **HUPCL** | 0002000 | Hang up on last close. |
| **CLOCAL** | 0004000 | Local line, else dial-up. |
| **CRTSCTS** | 0010000 | Enable RTS/CTS flow control. |
| **CIBAUD** | 03600000 | Input baud rate, if different from output rate. |

The **CBAUD** bits specify the baud rate. The zero baud rate, **B0,** is used to hang up the connection. If **B0** is specified, the modem control lines will cease to be asserted. Normally, this will disconnect the line. If the **CIBAUD** bits are not zero, they specify the input baud rate, with the **CBAUD** bits specifying the output baud rate; otherwise, the output and input baud rates are both specified by the **CBAUD** bits. The values for the **CIBAUD** bits are the same as the values for the **CBAUD** bits, shifted left **IBSHIFT** bits. For any particular hardware, impossible speed changes are ignored.

The **CSIZE** bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If **CSTOPB** is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stop bits are required.

If **PARENB** is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the **PARODD** flag specifies odd parity if set, otherwise even parity is used.

If **CREAD** is set, the receiver is enabled. Otherwise no characters will be received.

If **HUPCL** is set, the modem control lines for the port will be disconnected when the last process with the line open closes it or terminates.

If **CLOCAL** is set, a connection does not depend on the state of the modem status lines. Otherwise modem control is assumed.

If **CRTSCTS** is set, and the terminal has modem control lines associated with it, the Request To Send (RTS) modem control line will be raised, and output will occur only if the Clear To Send (CTS) modem status line is raised. If the CTS modem status line is lowered, output is suspended until CTS is raised. Some hardware may not support this function, and other hardware may not permit it to be disabled; in either of these cases, the state of the **CRTSCTS** flag is ignored.

The initial hardware control value after open is **B9600, CS7, CREAD, PARENB.**

**Local Modes**

The **c_lflag** field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

| | | |
|---|---|---|
| **ISIG** | 0000001 | Enable signals. |
| **ICANON** | 0000002 | Canonical input (erase and kill processing). |
| **XCASE** | 0000004 | Canonical upper/lower presentation. |
| **ECHO** | 0000010 | Enable echo. |
| **ECHOE** | 0000020 | Echo erase character as BS-SP-BS. |
| **ECHOK** | 0000040 | Echo NL after kill character. |
| **ECHONL** | 0000100 | Echo NL. |
| **NOFLSH** | 0000200 | Disable flush after interrupt or quit. |
| **TOSTOP** | 0000400 | Send **SIGTTOU** for background output. |
| **ECHOCTL** | 0001000 | Echo control characters as ^*char*, delete as ^?. |
| **ECHOPRT** | 0002000 | Echo erase character as character erased. |
| **ECHOKE** | 0004000 | BS-SP-BS erase entire line on line kill. |
| **FLUSHO** | 0040000 | Output is being flushed. |
| **PENDIN** | 0100000 | Retype pending input at next read or input character. |

If **ISIG** is set, each input character is checked against the special control characters **INTR**, **QUIT**, and **SUSP**. If an input character matches one of these control characters, the function associated with that character is performed. If **ISIG** is not set, no checking is done. Thus these special input functions are possible only if **ISIG** is set.

If **ICANON** is set, canonical processing is enabled. This enables the erase, word erase, kill, and reprint edit functions, and the assembly of input characters into lines delimited by NL, **EOF**, **EOL**, and **EOL2**. If **ICANON** is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least **MIN** characters have been received or the timeout value **TIME** has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The time value represents tenths of seconds. See the *Non-canonical Mode Input Processing* section for more details.

If **XCASE** is set, and if **ICANON** is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

| *for*: | *use*: |
|---|---|
| ` | \' |
| \| | \! |
| ~ | \^ |
| { | \( |
| } | \) |
| \ | \\ |

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If **ECHO** is set, characters are echoed as received. If **ECHO** is not set, input characters are not echoed.

If **ECHOCTL** is not set, all control characters (characters with codes between 0 and 37 octal) are echoed as themselves. If **ECHOCTL** is set, all control characters other than ASCII TAB, ASCII NL, the **START** character, and the **STOP** character, are echoed as ^*X*, where *X* is the character given by adding 100 octal to the control character's code (so that the character with octal code 1 is echoed as "^**A**'), and the ASCII DEL

character, with code 177 octal, is echoed as '^?'.

When **ICANON** is set, the following echo functions are possible:

1. If **ECHO** and **ECHOE** are set, and **ECHOPRT** is not set, the **ERASE** and **WERASE** characters are echoed as one or more ASCII BS SP BS, which will clear the last character(s) from a CRT screen.
2. If **ECHO** and **ECHOPRT** are set, the first **ERASE** and **WERASE** character in a sequence echoes as a backslash (\) followed by the characters being erased. Subsequent **ERASE** and **WERASE** characters echo the characters being erased, in reverse order. The next non-erase character types a slash (/) before it is echoed.
3. If **ECHOKE** is set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by **ECHOE** and **ECHOPRT**).
4. If **ECHOK** is set, and **ECHOKE** is not set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note: an escape character (\) or an **LNEXT** character preceding the erase or kill character removes any special function.
5. If **ECHONL** is set, the NL character will be echoed even if **ECHO** is not set. This is useful for terminals set to local echo (so-called half duplex).
6. If **ECHOCTL** is not set, the **EOF** character is not echoed, unless it is escaped. Because EOT is the default **EOF** character, this prevents terminals that respond to EOT from hanging up. If **ECHOCTL** is set, the **EOF** character is echoed; if it is not escaped, after it is echoed, one backspace character is output if it is echoed as itself, and two backspace characters are echoed if it is echoed as *^X*.

If **NOFLSH** is set, the normal flush of the input and output queues associated with the **INTR**, **QUIT**, and **SUSP** characters will not be done.

If **TOSTOP** is set, the signal **SIGTTOU** is sent to a process that tries to write to its controlling terminal if it is not in the distinguished process group for that terminal. This signal normally stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring **SIGTTOU** signals are excepted and allowed to produce output.

If **FLUSHO** is set, data written to the terminal will be discarded. This bit is set when the **FLUSH** character is typed. A program can cancel the effect of typing the **FLUSH** character by clearing **FLUSHO**.

If **PENDIN** is set, any input that has not yet been read will be reprinted when the next character arrives as input.

The initial line-discipline control value is **ISIG, ICANON, ECHO.**

**Minimum and Timeout**

The **MIN** and **TIME** values are described above under **Non-canonical Mode Input Processing**. The initial value of **MIN** is 1, and the initial value of **TIME** is 0.

**Termio Structure**

The System V **termio** structure is used by other **ioctl( )** calls; it is defined by **<sys/termio.h>** as:

```
#define  NCC      8
struct   termio {
         unsigned short  c_iflag;     /* input modes */
         unsigned short  c_oflag;     /* output modes */
         unsigned short  c_cflag;     /* control modes */
         unsigned short  c_lflag;     /* local modes */
         char            c_line;      /* line discipline */
         unsigned char   c_cc[NCC];   /* control chars */
};
```

The special control characters are defined by the array **c_cc**. The relative positions for each function are as follows:

0 **VINTR**
1 **VQUIT**
2 **VERASE**
3 **VKILL**
4 **VEOF**
5 **VEOL**
6 **VEOL2**
7 reserved

The calls that use the **termio** structure only affect the flags and control characters that can be stored in the **termio** structure; all other flags and control characters are unaffected.

**Terminal Size**

The number of lines and columns on the terminal's display (or page, in the case of printing terminals) is specified in the **winsize** structure, defined by **<sys/termios.h>**. Several **ioctl()** system calls that fetch or change these parameters use this structure:

```
struct winsize {
        unsigned short  ws_row;     /* rows, in characters */
        unsigned short  ws_col;     /* columns, in characters */
        unsigned short  ws_xpixel;  /* horizontal size, pixels - not used */
        unsigned short  ws_ypixel;  /* vertical size, pixels - not used */
};
```

**Modem Lines**

On special files representing serial ports, the modem control lines supported by the hardware can be read and the modem status lines supported by the hardware can be changed. The following modem control and status lines may be supported by a device; they are defined by **<sys/termios.h>**:

| | | |
|---|---|---|
| **TIOCM_LE** | 0001 | line enable |
| **TIOCM_DTR** | 0002 | data terminal ready |
| **TIOCM_RTS** | 0004 | request to send |
| **TIOCM_ST** | 0010 | secondary transmit |
| **TIOCM_SR** | 0020 | secondary receive |
| **TIOCM_CTS** | 0040 | clear to send |
| **TIOCM_CAR** | 0100 | carrier detect |
| **TIOCM_RNG** | 0200 | ring |
| **TIOCM_DSR** | 0400 | data set ready |

**TIOCM_CD** is a synonym for **TIOCM_CAR**, and **TIOCM_RI** is a synonym for **TIOCM_RNG**.

Not all of these will necessarily be supported by any particular device; check the manual page for the device in question.

**IOCTLS**

The **ioctl()** calls supported by devices and STREAMS modules providing the **termios** interface are listed below. Some calls may not be supported by all devices or modules.

Unless otherwise noted for a specific **ioctl()** call, these functions are restricted from use by background processes. Attempts to perform these calls will cause the process group of the process performing the call to be sent a **SIGTTOU** signal. If the process is ignoring **SIGTTOU**, has **SIGTTOU** blocked, or is in the middle of process creation using **vfork()**, the process will be allowed to perform the call and the **SIGTTOU** signal will not be sent.

**TCGETS** The argument is a pointer to a **termios** structure. The current terminal parameters are fetched and stored into that structure. This call is allowed from a background process; however, the information may subsequently be changed by a foreground process.

**TCSETS** The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

**TCSETSW** The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

**TCSETSF** The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

**TCGETA** The argument is a pointer to a **termio** structure. The current terminal parameters are fetched, and those parameters that can be stored in a **termio** structure are stored into that structure. This call is allowed from a background process; however, the information may subsequently be changed by a foreground process.

**TCSETA** The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change is immediate.

**TCSETAW** The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

**TCSETAF** The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

**TCSBRK** The argument is an **int** value. Wait for the output to drain. If the argument is 0, then send a break (zero-valued bits for 0.25 seconds).

**TCXONC** Start/stop control. The argument is an **int** value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.

**TCFLSH** The argument is an **int** value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

**TIOCEXCL** The argument is ignored. Exclusive-use mode is turned on; no further opens are permitted until the file has been closed, or a **TIOCNXCL** is issued. The default on open of a terminal file is that exclusive use mode is off.

**TIOCNXCL** The argument is ignored. Exclusive-use mode is turned off.

**TIOCGPGRP** The argument is a pointer to an **int**. Set the value of that **int** to the process group ID of the distinguished process group associated with the terminal. This call is allowed from a background process; however, the information may subsequently be changed by a foreground process.

**TIOCSPGRP** The argument is a pointer to an **int**. Associate the process group whose process group ID is specified by the value of that **int** with the terminal. The new process group value must be in the range of valid process group ID values, or it must be zero ("no process group"). Otherwise, the error **EINVAL** is returned. If any processes exist with a process ID or process group ID that is the same as the new process group value, then those processes must have the same real or saved user ID as the real or effective user ID of the calling process or be descendants of the calling process, or the effective user ID of the current process must be super-user. Otherwise, the error **EPERM** is returned.

**TIOCOUTQ** The argument is a pointer to an **int**. Set the value of that **int** to the number of characters

in the output stream that have not yet been sent to the terminal. This call is allowed from a background process.

**TIOCSTI** The argument is a pointer to a **char**. Pretend that that character had been received as input.

**TIOCGWINSZ** The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is stored into that structure. This call is allowed from a background process.

**TIOCSWINSZ** The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is set from the values specified in that structure. If the new sizes are different from the old sizes, a **SIGWINCH** signal is sent to the process group of the terminal.

**TIOCMGET** The argument is a pointer to an **int**. The current state of the modem status lines is fetched and stored in the **int** pointed to by the argument. This call is allowed from a background process.

**TIOCMBIS** The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned on. The control lines whose bits are set in the argument are turned on; no other control lines are affected.

**TIOCMBIC** The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned off. The control lines whose bits are set in the argument are turned off; no other control lines are affected.

**TIOCMSET** The argument is a pointer to an **int** containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.

**SEE ALSO**

**csh**(1), **stty**(1V), **fork**(2), **ioctl**(2), **open**(2V), **read**(2V), **setpgrp(2V)**, **sigvec**(2), **vfork**(2), **tty(4)**, **getty**(8)

## NAME

tm – tapemaster 1/2 inch tape drive

## CONFIG — SUN-3 SYSTEM

**controller tm0 at vme16d16 ? csr 0xa0 priority 3 vector tmintr 0x60**
**controller tm1 at vme16d16 ? csr 0xa2 priority 3 vector tmintr 0x61**
**tape mt0 at tm0 drive 0 flags 1**
**tape mt0 at tm1 drive 0 flags 1**

## CONFIG — SUN-2 SYSTEM

**controller tm0 at mbio ? csr 0xa0 priority 3**
**controller tm0 at vme16 ? csr 0xa0 priority 3 vector tmintr 0x60**
**controller tm1 at mbio ? csr 0xa2 priority 3**
**controller tm1 at vme16 ? csr 0xa2 priority 3 vector tmintr 0x61**
**tape mt0 at tm0 drive 0 flags 1**
**tape mt0 at tm1 drive 0 flags 1**

## DESCRIPTION

The Tapemaster tape controller controls Pertec-interface 1/2'' tape drives such as the CDC Keystone, providing a standard tape interface to the device, see **mtio**(4).

## SEE ALSO

**mt**(1), **tar**(1), **ar**(4S), **mtio**(4)

## DIAGNOSTICS

**tm*n* : no response from ctlr.**
**tm*n*: error *n*** during config.
**mt*n* : not online.**
**mt*n* : no write ring.**
**tmgo: gate wasn't open. Controller lost synch.**
**tmintr: can't clear interrupts.**
**tm*n* : stray interrupts.**
**mt*n*: hard error bn=*n* er=%x.**
**mt*n* : lost interrupt.**

## BUGS

The Tapemaster controller does not provide for byte-swapping and the resultant system overhead prevents streaming transports from streaming.

If a non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

The system should remember which controlling terminal has the tape drive open and write error messages to that terminal rather than on the console.

NAME

ttcompat – V7 and 4BSD STREAMS compatibility module

CONFIG

None; included by default.

SYNOPSIS

**#include <sys/stream.h>**
**#include <sys/stropt.h>**

**ioctl(fd, I_PUSH, "ttcompat");**

DESCRIPTION

**ttcompat** is a STREAMS module that translates the **ioctl** calls supported by the older Version 7 and 4BSD terminal drivers into the **ioctl** calls supported by the **termio**(4) interface. All other messages pass through this module unchanged; the behavior of **read** and **write** calls is unchanged, as is the behavior of **ioctl** calls other than the ones supported by **ttcompat**.

Normally, this module is automatically pushed onto a stream when a terminal device is opened; it does not have to be explicitly pushed onto a stream. This module requires that the **termio** interface be supported by the modules and driver downstream. The TCGETS, TCSETS, and TCSETSF **ioctl** calls must be supported; if any information set or fetched by those **ioctl** calls is not supported by the modules and driver downstream, some of the V7/4BSD functions may not be supported. For example, if the CBAUD bits in the **c_cflag** field are not supported, the functions provided by the **sg_ispeed** and **sg_ospeed** fields of the **sgttyb** structure (see below) will not be supported. If the TCFLSH **ioctl** is not supported, the function provided by the TIOCFLUSH **ioctl** will not be supported. If the TCXONC **ioctl** is not supported, the functions provided by the TIOCSTOP and TIOCSTART **ioctl** calls will not be supported. If the TIOCMBIS and TIOCMBIC **ioctl** calls are not supported, the functions provided by the TIOCSDTR and TIOCCDTR **ioctl** calls will not be supported.

The basic **ioctl** calls use the **sgttyb** structure defined by **<sys/ioctl.h>**:

```
struct sgttyb {
        char    sg_ispeed;
        char    sg_ospeed;
        char    sg_erase;
        char    sg_kill;
        short   sg_flags;
};
```

The **sg_ispeed** and **sg_ospeed** fields describe the input and output speeds of the device, and reflect the values in the **c_cflag** field of the **termio** structure. The **sg_erase** and **sg_kill** fields of the argument structure specify the erase and kill characters respectively, and reflect the values in the VERASE and VKILL members of the **c_cc** field of the **termio** structure.

The **sg_flags** field of the argument structure contains several flags that determine the system's treatment of the terminal. They are mapped into flags in fields of the terminal state, represented by the **termio** structure.

Delay type 0 is always mapped into the equivalent delay type 0 in the **c_oflag** field of the **termio** structure. Other delay mappings are performed as follows:

| **sg_flags** | **c_oflag** |
|---|---|
| **BS1** | **BS1** |
| **FF1** | **VT1** |
| **CR1** | **CR2** |
| **CR2** | **CR3** |
| **CR3** | not supported |
| **TAB1** | **TAB1** |
| **TAB2** | **TAB2** |
| **XTABS** | **TAB3** |

| | |
|---|---|
| **NL1** | **ONLRET\|CR1** |
| **NL2** | **NL1** |

If previous **TIOCLSET** or **TIOCLBIS** **ioctl** calls have not selected **LITOUT** or **PASS8** mode, and if **RAW** mode is not selected, the **ISTRIP** flag is set in the **c_iflag** field of the **termio** structure, and the **EVENP** and **ODDP** flags control the parity of characters sent to the terminal and accepted from the terminal:

**0** Parity is not to be generated on output or checked on input; the character size is set to **CS8** and the **PARENB** flag is cleared in the **c_cflag** field of the **termio** structure.

**EVENP** Even parity characters are to be generated on output and accepted on input; the **INPCK** flag is set in the **c_iflag** field of the **termio** structure, the character size is set to **CS7** and the **PARENB** flag is set in the **c_cflag** field of the **termio** structure.

**ODDP** Odd parity characters are to be generated on output and accepted on input; the **INPCK** flag is set in the **c_iflag** field, the character size is set to **CS7** and the **PARENB** and **PARODD** flags are set in the **c_cflag** field of the **termio** structure.

**EVENP|ODDP**
Even parity characters are to be generated on output and characters of either parity are to be accepted on input; the **INPCK** flag is cleared in the **c_iflag** field, the character size is set to **CS7** and the **PARENB** flag is set in the **c_cflag** field of the **termio** structure.

The **RAW** flag disables all output processing (the **OPOST** flag in the **c_oflag** field, and the **XCASE** flag in the **c_lflag** field, are cleared in the **termio** structure) and input processing (all flags in the **c_iflag** field other than the **IXOFF** and **IXANY** flags are cleared in the **termio** structure). 8 bits of data, with no parity bit, are accepted on input and generated on output; the character size is set to **CS8** and the **PARENB** and **PARODD** flags are cleared in the **c_cflag** field of the **termio** structure. The signal-generating and line-editing control characters are disabled by clearing the **ISIG** and **ICANON** flags in the **c_lflag** field of the **termio** structure.

The **CRMOD** flag turn input RETURN characters into NEWLINE characters, and output and echoed NEWLINE characters to be output as a RETURN followed by a LINEFEED. The **ICRNL** flag in the **c_iflag** field, and the **OPOST** and **ONLCR** flags in the **c_oflag** field, are set in the **termio** structure.

The **LCASE** flag maps upper-case letters in the ASCII character set to their lower-case equivalents on input (the **IUCLC** flag is set in the **c_iflag** field), and maps lower-case letters in the ASCII character set to their upper-case equivalents on output (the **OLCUC** flag is set in the **c_oflag** field). Escape sequences are accepted on input, and generated on output, to handle certain ASCII characters not supported by older terminals (the **XCASE** flag is set in the **c_lflag** field).

Other flags are directly mapped to flags in the **termio** structure:

| | |
|---|---|
| **sg_flags** | flags in **termio** structure |
| **CBREAK** | complement of **ICANON** in **c_lflag** field |
| **ECHO** | **ECHO** in **c_lflag** field |
| **TANDEM** | **IXOFF** in **c_iflag** field |

Another structure associated with each terminal specifies characters that are special in both the old Version 7 and the newer 4BSD terminal interfaces. The following structure is defined by **<sys/ioctl.h>**:

```
struct tchars {
        char    t_intrc;        /* interrupt */
        char    t_quitc;        /* quit */
        char    t_startc;       /* start output */
        char    t_stopc;        /* stop output */
        char    t_eofc;         /* end-of-file */
        char    t_brkc;         /* input delimiter (like nl) */
};
```

The characters are mapped to members of the **c_cc** field of the **termio** structure as follows:

| **tchars** | **c_cc** index |
|---|---|
| **t_intrc** | **VINTR** |
| **t_quitc** | **VQUIT** |
| **t_startc** | **VSTART** |
| **t_stopc** | **VSTOP** |
| **t_eofc** | **VEOF** |
| **t_brkc** | **VEOL** |

Also associated with each terminal is a local flag word, specifying flags supported by the new 4BSD terminal interface. Most of these flags are directly mapped to flags in the **termio** structure:

| local flags | flags in **termio** structure |
|---|---|
| **LCRTBS** | not supported |
| **LPRTERA** | **ECHOPRT** in the **c_lflag** field |
| **LCRTERA** | **ECHOE** in the **c_lflag** field |
| **LTILDE** | not supported |
| **LTOSTOP** | **TOSTOP** in the **c_lflag** field |
| **LFLUSHO** | **FLUSHO** in the **c_lflag** field |
| **LNOHANG** | **CLOCAL** in the **c_cflag** field |
| **LCRTKIL** | **ECHOKE** in the **c_lflag** field |
| **LCTLECH** | **CTLECH** in the **c_lflag** field |
| **LPENDIN** | **PENDIN** in the **c_lflag** field |
| **LDECCTQ** | complement of **IXANY** in the **c_iflag** field |
| **LNOFLSH** | **NOFLSH** in the **c_lflag** field |

Another structure associated with each terminal is the **ltchars** structure which defines control characters for the new 4BSD terminal interface. Its structure is:

```
struct ltchars {
        char    t_suspc;        /* stop process signal */
        char    t_dsuspc;       /* delayed stop process signal */
        char    t_rprntc;       /* reprint line */
        char    t_flushc;       /* flush output (toggles) */
        char    t_werasc;       /* word erase */
        char    t_lnextc;       /* literal next character */
};
```

The characters are mapped to members of the **c_cc** field of the **termio** structure as follows:

| **ltchars** | **c_cc** index |
|---|---|
| **t_suspc** | **VSUSP** |
| **t_dsuspc** | **VDSUSP** |
| **t_rprntc** | **VREPRINT** |
| **t_flushc** | **VDISCARD** |
| **t_werasc** | **VWERASE** |
| **t_lnextc** | **VLNEXT** |

IOCTLS

**ttcompat** responds to the following **ioctl** calls. All others are passed to the module below.

TIOCGETP The argument is a pointer to an **sgttyb** structure. The current terminal state is fetched; the appropriate characters in the terminal state are stored in that structure, as are the input and output speeds. The values of the flags in the **sg_flags** field are derived from the flags in the terminal state and stored in the structure.

**TIOCSETP** The argument is a pointer to an **sgttyb** structure. The appropriate characters and input and output speeds in the terminal state are set from the values in that structure, and the flags in the terminal state are set to match the values of the flags in the **sg_flags** field of that structure. The state is changed with a **TCSETSF** *ioctl*, so that the interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

**TIOCSETN** The argument is a pointer to an **sgttyb** structure. The terminal state is changed as **TIOCSETP** would change it, but a **TCSETS ioctl** is used, so that the interface neither delays nor discards input.

**TIOCHPCL** The argument is ignored. The **HUPCL** flag is set in the **c_cflag** word of the terminal state.

**TIOCFLUSH** The argument is a pointer to an **int** variable. If its value is zero, all characters waiting in input or output queues are flushed. Otherwise, the value of the **int** is treated as the logical OR of the **FREAD** and **FWRITE** flags defined by **<sys/file.h>**; if the **FREAD** bit is set, all characters waiting in input queues are flushed, and if the **FWRITE** bit is set, all characters waiting in output queues are flushed.

**TIOCSBRK** The argument is ignored. The break bit is set for the device.

**TIOCCBRK** The argument is ignored. The break bit is cleared for the device.

**TIOCSDTR** The argument is ignored. The Data Terminal Ready bit is set for the device.

**TIOCCDTR** The argument is ignored. The Data Terminal Ready bit is cleared for the device.

**TIOCSTOP** The argument is ignored. Output is stopped as if the **STOP** character had been typed.

**TIOCSTART** The argument is ignored. Output is restarted as if the **START** character had been typed.

**TIOCGETC** The argument is a pointer to an **tchars** structure. The current terminal state is fetched, and the appropriate characters in the terminal state are stored in that structure.

**TIOCSETC** The argument is a pointer to an **tchars** structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

**TIOCLGET** The argument is a pointer to an **int**. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state and stored in the **int** pointed to by the argument.

**TIOCLBIS** The argument is a pointer to an **int** whose value is a mask containing flags to be set in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are set, and the flags in the terminal state are set to match the new value of the local flags word.

**TIOCLBIC** The argument is a pointer to an **int** whose value is a mask containing flags to be cleared in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are cleared, and the flags in the terminal state are set to match the new value of the local flags word.

**TIOCLSET** The argument is a pointer to an **int** containing a new set of local flags. The flags in the terminal state are set to match the new value of the local flags word.

**TIOCGLTC** The argument is a pointer to an **ltchars** structure. The values of the appropriate characters in the terminal state are stored in that structure.

**TIOCSLTC** The argument is a pointer to an **ltchars** structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.

**SEE ALSO**

**ioctl**(2), **termio**(4)

## NAME

tty – controlling terminal interface

## DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the controlling terminal of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

## IOCTLS

In addition to the **ioctl**s supported by the device that **tty** refers to, the following **ioctl** is supported:

**TIOCNOTTY** Detach the current process from its controlling terminal, and remove it from its current process group, without attaching it to a new process group (that is, set its process group ID to zero). This **ioctl** call only works on file descriptors connected to **/dev/tty**; this is used by daemon processes when they are invoked by a user at a terminal. The process attempts to open **/dev/tty**; if the open succeeds, it detaches itself from the terminal by using **TIOCNOTTY**, while if the open fails, it is obviously not attached to a terminal and does not need to detach itself.

## FILES

**/dev/tty**

## SEE ALSO

**termio**(4)

## NAME

udp – Internet User Datagram Protocol

## SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
```

## DESCRIPTION

UDP is a simple, unreliable datagram protocol which is used to support the **SOCK_DGRAM** abstraction for the Internet protocol family. It is layered directly above the Internet Protocol (IP). UDP sockets are connectionless, and are normally used with the **sendto**, **sendmsg**, **recvfrom**, and **recvmsg** system calls (see **send**(2) and **recv**(2)). If the **connect**(2) system call is used to fix the destination for future packets, then the **recv**(2) or **read**(2V) and **send**(2) or **write**(2V) system calls may be used.

UDP address formats are identical to those used by the Transmission Control Protocol (TCP). Like TCP, UDP uses a port number along with an IP address to identify the endpoint of communication. Note: the UDP port number space is separate from the TCP port number space (that is, a UDP port may not be "connected" to a TCP port). The **bind**(2) system call can be used to set the local address and port number of a UDP socket. The local IP address may be left unspecified in the **bind** call by using the special value **INADDR_ANY**. If the **bind** call is not done, a local IP address and port number will be assigned to each packet as it is sent. Broadcast packets may be sent (assuming the underlying network supports this) by using a reserved "broadcast address"; this address is network interface dependent. Broadcasts may only be sent by the super-user.

Options at the IP level may be used with UDP; see **ip**(4P).

There are a variety of ways that a UDP packet can be lost or discarded, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is in error, the packet will be dropped with no indication given to the user. A queue of received packets is provided for each UDP socket. This queue has a limited capacity. Arriving datagrams which will not fit within its *high-water* capacity are silently discarded.

UDP processes Internet Control Message Protocol (ICMP) error messages received in response to UDP packets it has sent. See **icmp**(4P). ICMP "source quench" messages are ignored. ICMP "destination unreachable," "time exceeded" and "parameter problem" messages disconnect the socket from its peer so that subsequent attempts to send packets using that socket will return an error. UDP will not guarantee that packets are delivered in the order they were sent. As well, duplicate packets may be generated in the communication process.

## ERRORS

A socket operation may fail if:

| | |
|---|---|
| EISCONN | A **connect** operation was attempted on a socket on which a **connect** operation had already been performed, and the socket could not be successfully disconnected before making the new connection. |
| EISCONN | A **sendto** or **sendmsg** operation specifying an address to which the message should be sent was attempted on a socket on which a **connect** operation had already been performed. |
| ENOTCONN | A **send** or **write** operation, or a **sendto** or **sendmsg** operation not specifying an address to which the message should be sent, was attempted on a socket on which a **connect** operation had not already been performed. |
| EADDRINUSE | A **bind** operation was attempted on a socket with a network address/port pair that has already been bound to another socket. |
| EADDRNOTAVAIL | A **bind** operation was attempted on a socket with a network address for which no network interface exists. |

| | |
|---|---|
| EINVAL | A **sendmsg** operation with a non-NULL **msg_accrights** was attempted. |
| EACCES | A **bind** operation was attempted with a "reserved" port number and the effective user ID of the process was not super-user. |
| ENOBUFS | The system ran out of memory for internal data structures. |

SEE ALSO

**bind**(2), **connect**(2), **read**(2V), **recv**(2), **send**(2), **write**(2V), **icmp**(4P), **inet**(4F), **ip**(4P), **tcp**(4P)

Postel, Jon, *User Datagram Protocol*, RFC 768, Network Information Center, SRI International, Menlo Park, Calif., August 1980. (Sun 800-1054-01)

BUGS

**SIOCSHIWAT** and **SIOCGHIWAT** ioctl's to set and get the high water mark for the socket queue, and so that it can be changed from 2048 bytes to be larger or smaller, have been defined (in **<sys/ioctl.h>**) but not implemented.

Something sensible should be done with ICMP source quench error messages if the socket is bound to a peer socket.

## NAME

vp – Ikon 10071-5 Versatec parallel printer interface

## CONFIG — SUN-2 SYSTEM

**device vp0 at mbio ? csr**

## DESCRIPTION

This Sun interface to the Versatec printer/plotter is supported by the Ikon parallel interface board, a word DMA device, which is output only.

The Versatec is normally handled by the line printer spooling system and should not be accessed by the user directly.

Opening the device **/dev/vp0** may yield one of two errors: ENXIO indicates that the device is already in use; EIO indicates that the device is offline.

The printer operates in either print or plot mode. To set the printer into plot mode you should include **<vcmd.h>** and use the **ioctl**(2) call

```
ioctl(f, VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[ ] = { VPLOT, 0, 0 };
```

When going back into print mode from plot mode you normally eject paper by sending it an EOT after putting into print mode:

```
int prtmd[ ] = { VPRINT, 0, 0 };
...
fflush (vp);
f = fileno (vp);
ioctl(f, VSETSTATE, prtmd);
write(f, "\04", 1);
```

## FILES

**/dev/vp0**

## SEE ALSO

**ioctl**(2)

## BUGS

If you use the standard I/O library on the Versatec, be sure to explicitly set a buffer using *setbuf*, since the library will not use buffered output by default, and will run very slowly.

Writes must start on even byte boundaries and be an even number of bytes in length.

## NAME

vpc – Systech VPC-2200 Versatec printer/plotter and Centronics printer interface

## CONFIG — SUN-2 SYSTEM

**device vpc0 at mbio ? csr 0x480 priority 2**
**device vpc1 at mbio ? csr 0x500 priority 2**

## DESCRIPTION

This Sun interface to the Versatec printer/plotter and to Centronics printers is supported by the Systech parallel interface board, an output-only byte-wide DMA device. The device has one channel for Versatec devices and one channel for Centronics devices, with an optional long lines interface for Versatec devices.

Devices attached to this interface are normally handled by the line printer spooling system and should not be accessed by the user directly.

Opening the device **/dev/vpc0** or **/dev/lp0** may yield one of two errors: ENXIO indicates that the device is already in use; EIO indicates that the device is offline.

The Versatec printer/plotter operates in either print or plot mode. To set the printer into plot mode you should include **<vcmd.h>** and use the **ioctl**(2) call:

```
ioctl(f, VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[ ] = { VPLOT, 0, 0 };
```

When going back into print mode from plot mode you normally eject paper by sending it an EOT after putting into print mode:

```
int prtmd[ ] = { VPRINT, 0, 0 };
...
fflush (vpc);
f = fileno(vpc);
ioctl(f, VSETSTATE, prtmd);
write(f, "\04", 1);
```

## FILES

**/dev/vpc0**
**/dev/lp0**

## SEE ALSO

**ioctl**(2)

## BUGS

If you use the standard I/O library on the Versatec, be sure to explicitly set a buffer using *setbuf*, since the library will not use buffered output by default, and will run very slowly.

## NAME

win – Sun window system

## CONFIG

**pseudo-device win***number*
**pseudo-device dtop***number*

## DESCRIPTION

The **win** pseudo-device accesses the system drivers supporting the Sun window system. *number*, in the device description line above, indicates the maximum number of windows supported by the system. *number* is set to 128 in the GENERIC system configuration file used to generate the kernel used in Sun systems as they are shipped. The *dtop* pseudo-device line indicates the number of separate "desktops" (frame buffers) that can be actively running the Sun window system at once. In the GENERIC file, this number is set to 4.

Each window in the system is represented by a **/dev/win*** device. The windows are organized as a tree with windows being subwindows of their parents, and covering/covered by their siblings. Each window has a position in the tree, a position on a display screen, an input queue, and information telling what parts of it are exposed.

The window driver multiplexes keyboard and mouse input among the several windows, tracks the mouse with a cursor on the screen, provides each window access to information about what parts of it are exposed, and notifies the manager process for a window when the exposed area of the window changes so that the window may repair its display.

Full information on the window system functions is given in the *SunView 1 System Programmer's Guide*.

## FILES

**/dev/win[0-9]**
**/dev/win[0-9][0-9]**

## SEE ALSO

*SunView 1 System Programmer's Guide*

## NAME

xd – Disk driver for Xylogics 7053 SMD Disk Controller

## CONFIG — SUN-3 SYSTEM

**controller xdc0 at vme16d32 ? csr 0xee80 priority 2 vector xdintr 0x44**
**controller xdc1 at vme16d32 ? csr 0xee90 priority 2 vector xdintr 0x45**
**controller xdc2 at vme16d32 ? csr 0xeea0 priority 2 vector xdintr 0x46**
**controller xdc3 at vme16d32 ? csr 0xeeb0 priority 2 vector xdintr 0x47**
**disk xd0 at xdc0 drive 0**
**disk xd1 at xdc0 drive 1**
**disk xd2 at xdc0 drive 2**
**disk xd3 at xdc0 drive 3**
**disk xd4 at xdc1 drive 0**
**disk xd5 at xdc1 drive 1**
**disk xd6 at xdc1 drive 2**
**disk xd7 at xdc1 drive 3**
**disk xd8 at xdc2 drive 0**
**disk xd9 at xdc2 drive 1**
**disk xd10 at xdc2 drive 2**
**disk xd11 at xdc2 drive 3**
**disk xd12 at xdc3 drive 0**
**disk xd13 at xdc3 drive 1**
**disk xd14 at xdc3 drive 2**
**disk xd15 at xdc3 drive 3**

The four **controller** lines given in the synopsis section above specify the first, second, third, and fourth Xylogics 7053 SMD disk controller in a Sun system.

## DESCRIPTION

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, and so on. The standard device names begin with **xd** followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk using the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call usually results in only one I/O operation; therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra **r**.

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise **directory**(3) calls should specify a multiple of 512 bytes.

If **flags 0x1** is specified, the overlapped seeks feature for that drive is turned off. Note: to be effective, the flag must be set on all drives for a specific controller. This action is necessary for controllers with older firmware, which have bugs preventing overlapped seeks from working properly.

## DISK SUPPORT

This driver handles all SMD drives by reading a label from sector 0 of the drive which describes the disk geometry and partitioning.

The **xd?a** partition is normally used for the root file system on a disk, the xd?b partition as a paging area, and the **xd?c** partition for pack-pack copying (it normally maps the entire disk). The rest of the disk is normally the **xd?g** partition.

## FILES

| | |
|---|---|
| **/dev/xd[0-7][a-h]** | block files |
| **/dev/rxd[0-7][a-h]** | raw files |

SEE ALSO

**lseek**(2), **read**(2V), **write**(2V), **directory**(3), **dkio**(4S)

DIAGNOSTICS

**xdc*n*: self test error**

Self test error in controller, see the Maintenance and Reference Manual.

**xd*n*: unable to read bad sector**

The bad sector forwarding information for the disk could not be read.

**xd*n*: initialization failed**

The drive could not be successfully initialized.

**xd*n*: unable to read label**

The drive geometry/partition table information could not be read.

**xd*n*: Corrupt label**

The geometry/partition label checksum was incorrect.

**xd*n*: offline**

A drive ready status is no longer detected, so the unit has been logically removed from the system. If the drive ready status is restored, the unit will automatically come back online the next time it is accessed.

**xd*nc*:** *cmd how* (*msg*) **blk** *#n abs blk #n*

A command such as read or write encountered an error condition (*how*): either it *failed*, the controller was *reset*, the unit was *restored*, or an operation was *retry*'ed. The *msg* is derived from the error number given by the controller, indicating a condition such as "drive not ready(rq, "sector not found" or "disk write protected". The *blk #* is the sector in error relative to the beginning of the partition involved. The *abs blk #* is the absolute block number of the sector in error. Some fields of the error message may be missing since the information is not always available.

BUGS

In raw I/O **read**(2V) and **write**(2V) truncate file offsets to 512-byte block boundaries, and **write**(2V) scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, **read**(2V), **write**(2V) and **lseek**(2) should always deal in 512-byte multiples.

Older revisions of the firmware do not properly support overlapped seeks. This will only affect systems with multiple disks on a single controller. If a large number of "zero sector count" errors appear, you should use the **flags** field to disable overlapped seeks.

## NAME

xt – Xylogics 472 1/2 inch tape controller

## CONFIG — SUN-3 SYSTEM

**controller xtc0 at vme16d16 ? csr 0xee60 priority 3 vector xtintr 0x64**
**controller xtc1 at vme16d16 ? csr 0xee68 priority 3 vector xtintr 0x65**
**tape xt0 at xtc0 drive 0 flags 1**
**tape xt1 at xtc1 drive 0 flags 1**

## CONFIG — SUN-2 SYSTEM

**controller xtc0 at mbio ? csr 0xee60 priority 3**
**controller xtc0 at vme16 ? csr 0xee60 priority 3 vector xtintr 0x64**
**controller xtc1 at mbio ? csr 0xee68 priority 3**
**controller xtc1 at vme16 ? csr 0xee68 priority 3 vector xtintr 0x65**
**tape xt0 at xtc0 drive 0 flags 1**
**tape xt1 at xtc1 drive 0 flags 1**

## DESCRIPTION

The Xylogics 472 tape controller controls Pertec-interface 1/2'' tape drives such as the CDC Keystone III, providing a standard tape interface to the device, see **mtio**(4). This controller is used to support high speed or high density drives, which are not supported effectively by the older TapeMaster controller (see **tm**(4S)).

The flags field is used to control remote density select operation: a 0 specifies no remote density selection is to be attempted, a 1 specifies that the Pertec density-select line is used to toggle between high and low density; a 2 specifies that the Pertec speed-select line is used to toggle between high and low density. The default is 1, which is appropriate for the CDC Keystone III (92185) and the Telex 9250. In no case will the controller select among more than 2 densities.

## SEE ALSO

**mt**(1), **tar**(1), **mtio**(4), **tm**(4S)

NAME

x

CONFIG -

c
c
d
d
d
d

T
4

CONFIG -

c
c
c
c
d
d
d
d

T
S
g

DESCRIP

F
1
tl
r

T
t
t
i
t

I
s

I
fl
fi

DISK SUP

T
g

T
a
r

FILES

/
/

NAME

zero – source of zeroes

SYNOPSIS

None; included with standard system.

DESCRIPTION

A zero special file is a source of zeroed unnamed memory.

Reads from a zero special file always return a buffer full of zeroes. The file is of

Writes to a zero special file are always successful, but the data written is ignored.

Mapping a zero special file creates a zero-initialized unnamed memory object length of the mapping and rounded up to the nearest page size as returned by processes can share such a zero special file object provided a common anc **MAP_SHARED**.

FILES

**/dev/zero**

SEE ALSO

**fork**(2), **getpagesize**(2), **mmap**(2)

NAME

zs – Zilog 8530 SCC serial communications driver

CONFIG — SUN-3 SYSTEM

**device zs0 at obio ? csr 0x20000 flags 3 priority 3**
**device zs1 at obio ? csr 0x00000 flags 0x103 priority 3**

CONFIG — SUN-2 SYSTEM

**device zs0 at virtual ? csr 0xeec800 flags 3 priority 3**
**device zs1 at virtual ? csr 0xeec000 flags 0x103 priority 3**
**device zs2 at mbmem ? csr 0x80800 flags 3 priority 3**
**device zs3 at mbmem ? csr 0x81000 flags 3 priority 3**
**device zs4 at mbmem ? csr 0x84800 flags 3 priority 3**
**device zs5 at mbmem ? csr 0x85000 flags 3 priority 3**

CONFIG — Sun386i SYSTEM

**device zs0 at obmem ? csr 0xFC000000 flags 3 irq 9 priority 6**
**device zs1 at obmem ? csr 0xA0000020 flags 0x103 irq 9 priority 6**

SYNOPSIS

**#include <fcntl.h>**
**#include <sys/termios.h>**
**open("/dev/tty*n*", mode);**
**open("/dev/ttyd*n*", mode);**
**open("/dev/cua*n*", mode);**

DESCRIPTION

The Zilog 8530 provides 2 serial communication ports with full modem control in asynchronous mode. Each port supports those **termio**(4) device control functions specified by flags in the **c_cflag** word of the **termios** structure and by the IGNBRK, IGNPAR, PARMRK, or INPCK flags in the **c_iflag** word of the **termios** structure are performed by the **zs** driver. All other **termio**(4) functions must be performed by STREAMS modules pushed atop the driver; when a device is opened, the **ldterm**(4M) and **ttcompat**(4M) STREAMS modules are automatically pushed on top of the stream, providing the standard **termio**(4) interface.

Of the synopsis lines above, the line for **zs0** specifies the serial I/O port(s) provided by the CPU board, the line for **zs1** specifies the Video Board ports (which are used for keyboard and mouse), the lines for **zs2** and **zs3** specify the first and second ports on the first SCSI board in a system, and those for **zs4** and **zs5** specify the first and second ports provided by the second SCSI board in a system, respectively.

Bit *i* of **flags** may be specified to say that a line is not properly connected, and that the line *i* should be treated as hard-wired with carrier always present. Thus specifying **flags 0x2** in the specification of **zs0** would treat line **/dev/ttyb** in this way.

Minor device numbers in the range 0 – 11 correspond directly to the normal tty lines and are named **/dev/ttya** and **/dev/ttyb** for the two serial ports on the CPU board and **/dev/ttys*n*** for the ports on the SCSI boards; *n* is 0 or 1 for the ports on the first SCSI board, and 2 or 3 for the ports on the second SCSI board.

To allow a single tty line to be connected to a modem and used for both incoming and outgoing calls, a special feature, controlled by the minor device number, has been added. Minor device numbers in the range 128 – 139 correspond to the same physical lines as those above (that is, the same line as the minor device number minus 128).

A dial-in line has a minor device in the range 0 – 11 and is conventionally renamed **/dev/ttyd*n***, where *n* is a number indicating which dial-in line it is (so that **/dev/ttyd0** is the first dial-in line), and the dial-out line corresponding to that dial-in line has a minor device number 128 greater than the minor device number of the dial-in line and is conventionally named **/dev/cua*n***, where *n* is the number of the dial-in line.

The **/dev/cua*n*** lines are special in that they can be opened even when there is no carrier on the line. Once a **/dev/cua*n*** line is opened, the corresponding tty line can not be opened until the **/dev/cua*n*** line is closed; a blocking open will wait until the **/dev/cua*n*** line is closed (which will drop Data Terminal Ready, after which Carrier Detect will usually drop as well) and carrier is detected again, and a non-blocking open will return an error. Also, if the **/dev/ttyd*n*** line has been opened successfully (usually only when carrier is recognized on the modem) the corresponding **/dev/cua*n*** line can not be opened. This allows a modem to be attached to e.g. **/dev/ttyd0** (renamed from **/dev/ttya**) and used for dialin (by enabling the line for login in **/etc/ttytab**) and also used for dialout (by **tip**(1C) or **uucp**(1C)) as **/dev/cua0** when no one is logged in on the line. Note: the bit in the **flags** word in the configuration file (see above) must be zero for this line, which enables hardware carrier detection.

## IOCTLS

The standard set of **termio ioctl()** calls are supported by **zs**.

If the CRTSCTS flag in the **c_cflag** is set, output will be generated only if CTS is high; if CTS is low, output will be frozen. If the CRTSCTS flag is clear, the state of CTS has no effect. Breaks can be generated by the **TCSBRK**, **TIOCSBRK**, and **TIOCCBRK ioctl()** calls. The modem control lines **TIOCM_CAR**, **TIOCM_CTS**, **TIOCM_RTS**, and **TIOCM_DTR** are provided.

The input and output line speeds may be set to any of the speeds supported by **termio**. The speeds cannot be set independently; when the output speed is set, the input speed is set to the same speed.

## ERRORS

An **open()** will fail if:

| | |
|---|---|
| ENXIO | The unit being opened does not exist. |
| EBUSY | The dial-out device is being opened and the dial-in device is already open, or the dial-in device is being opened with a no-delay open and the dial-out device is already open. |
| EBUSY | The unit has been marked as exclusive-use by another process with a **TIOCEXCL ioctl()** call. |
| EINTR | The open was interrupted by the delivery of a signal. |

## FILES

| | |
|---|---|
| **/dev/tty{a,b,s[0-3]}** | hardwired tty lines |
| **/dev/ttyd[0-9a-f]** | dialin tty lines |
| **/dev/cua[0-9a-f]** | dialout tty lines |

## SEE ALSO

**tip**(1C), **uucp**(1C), **mcp**(4S), **mti**(4S), **termio**(4), **ldterm**(4M), **ttcompat**(4M)

## DIAGNOSTICS

**zs*n c*: silo overflow.**
The 8530 character input silo overflowed before it could be serviced.

**zs*n c*: ring buffer overflow.**
The driver's character input ring buffer overflowed before it could be serviced.

NAME

intro – file formats used or read by various programs

DESCRIPTION

This section describes formats of files used by various programs.

LIST OF FILE FORMATS

| Name | Appears on Page | Description |
|---|---|---|
| **a.out** | **a.out**(5) | assembler and link editor output format |
| **acct** | **acct**(5) | execution accounting file |
| **addresses** | **aliases**(5) | addresses and aliases for sendmail(8) |
| **aliases** | **aliases**(5) | addresses and aliases for sendmail(8) |
| **ar** | **ar**(5) | archive (library) file format |
| **audit.log** | **audit.log**(5) | the security audit trail file |
| **audit_control** | **audit_control**(5) | control information for system audit daemon |
| **audit_data** | **audit_data**(5) | current information on audit daemon |
| **auto.home** | **auto.home**(5) | automount map for home directories |
| **auto.vol** | **auto.vol**(5) | automount map for volumes |
| **bar** | **bar**(5) | tape archive file format |
| **bootparams** | **bootparams**(5) | boot parameter data base |
| **COFF** | **coff**(5) | common assembler and link editor output |
| **core** | **core**(5) | format of memory image file |
| **cpio** | **cpio**(5) | format of cpio archive |
| **crontab** | **crontab**(5) | table of times to run periodic jobs |
| **defaults** | **defaults**(5) | default specifications for SunView |
| **dir** | **dir**(5) | format of directories |
| **dump** | **dump**(5) | incremental dump format |
| **dumpdates** | **dump**(5) | incremental dump format |
| **environ** | **environ**(5V) | user environment |
| **ethers** | **ethers**(5) | Ethernet address to hostname database or YP domain |
| **exports** | **exports**(5) | directories to export to NFS clients |
| **fcntl** | **fcntl**(5) | file control options |
| **forward** | **aliases**(5) | addresses and aliases for sendmail(8) |
| **fs** | **fs**(5) | format of a 4.2 (ufs) file system volume |
| **fspec** | **fspec**(5) | format specification in text files |
| **fstab** | **fstab**(5) | static filesystem mounting table, mounted filesystems table |
| **ftpusers** | **ftpusers**(5) | list of users prohibited by ftp |
| **gettytab** | **gettytab**(5) | terminal configuration data base |
| **group.adjunct** | **group**(5) | group security data file |
| **group** | **group**(5) | group file |
| **help** | **help**(5) | help file format |
| **help_viewer** | **help_viewer**(5) | help viewer file format |
| **hosts.equiv** | **hosts**(5) | list of trusted hosts |
| **hosts** | **hosts**(5) | host name data base |
| **inetd.conf** | **inetd.conf**(5) | Internet servers database |
| **inode** | **fs**(5) | format of a 4.2 (ufs) file system volume |
| **internat** | **internat**(5) | key mapping table for internationalization |
| **ipalloc.netrange** | **ipalloc.netrange**(5) | range of addresses to allocate |
| **lastlog** | **utmp**(5) | login records |
| **magic** | **magic**(5) | file command's magic number file |
| **mtab** | **fstab**(5) | static filesystem mounting table, mounted filesystems table |
| **mtab** | **mtab**(5) | mounted file system table |
| **netgroup** | **netgroup**(5) | list of network groups |
| **netmasks** | **netmasks**(5) | network mask data base |

| | | |
|---|---|---|
| **netrc** | **netrc**(5) | file for ftp(1) remote login data |
| **networks** | **networks**(5) | network name data base |
| **passwd.adjunct** | **passwd**(5) | user security data file |
| **passwd** | **passwd**(5) | password file |
| **phones** | **phones**(5) | remote host phone number data base |
| **plot** | **plot**(5) | graphics interface |
| **policies** | **policies**(5) | network administration policies |
| **printcap** | **printcap**(5) | printer capability data base |
| **protocols** | **protocols**(5) | protocol name data base |
| **publickey** | **publickey**(5) | publickey database |
| **queuedefs** | **queuedefs**(5) | at/batch/cron queue description file |
| **rasterfile** | **rasterfile**(5) | Sun's file format for raster images |
| **remote** | **remote**(5) | remote host description file |
| **rgb** | **rgb**(5) | available colors (by name) for coloredit |
| **rootmenu** | **rootmenu**(5) | root menu specification for SunView |
| **rpc** | **rpc**(5) | rpc program number data base |
| **sccsfile** | **sccsfile**(5) | format of SCCS file |
| **services** | **services**(5) | Internet services and aliases |
| **sm.bak** | **sm**(5) | in.statd directory and file structures |
| **sm.bak** | **statmon**(5) | statd directories and file structures |
| **sm.state** | **sm**(5) | in.statd directory and file structures |
| **sm** | **sm**(5) | in.statd directory and file structures |
| **sm** | **statmon**(5) | statd directories and file structures |
| **state** | **statmon**(5) | statd directories and file structures |
| **sunview** | **sunview**(5) | initialization file for SunView |
| **syslog.conf** | **syslog.conf**(5) | configuration file for syslogd system log daemon |
| **tar** | **tar**(5) | tape archive file format |
| **term** | **term**(5) | terminal driving tables for nroff |
| **term** | **term**(5V) | format of compiled term file |
| **termcap** | **termcap**(5) | terminal capability data base |
| **terminfo** | **terminfo**(5V) | terminal capability data base |
| **textswrc** | **textswrc**(5) | initialization file for SunView text windows |
| **toc** | **toc**(5) | table of contents of optional clusters |
| **translate** | **translate**(5) | input and output files for system message translation |
| **ttys** | **ttys**(5) | terminal initialization data |
| **ttytab** | **ttytab**(5) | terminal initialization data |
| **ttytype** | **ttytype**(5) | data base of terminal types by port |
| **types** | **types**(5) | primitive system data types |
| **tzfile** | **tzfile**(5) | time zone information |
| **updaters** | **updaters**(5) | configuration file for YP updating |
| **utmp** | **utmp**(5) | login records |
| **uuencode** | **uuencode**(5) | format of an encoded uuencode file |
| **vfont** | **vfont**(5) | font formats |
| **vgrindefs** | **vgrindefs**(5) | vgrind's language definition data base |
| **wtmp** | **utmp**(5) | login records |
| **xtab** | **exports**(5) | directories to export to NFS clients |
| **ypfiles** | **ypfiles**(5) | the Yellow Pages database and directory structure |

## NAME

a.out – assembler and link editor output format

## SYNOPSIS

```
#include <a.out.h>
#include <stab.h>
#include <nlist.h>
```

## AVAILABILITY

Sun-2, Sun-3, and Sun-4 systems only. For Sun386i systems refer to **coff**(5).

## DESCRIPTION

**a.out** is the output format of the assembler **as**(1) and the link editor **ld**(1). The link editor makes **a.out** executable files.

A file in **a.out** format consists of: a header, the program text, program data, text and data relocation information, a symbol table, and a string table (in that order). In the header, the sizes of each section are given in bytes. The last three sections may be absent if the program was loaded with the **–s** option of **ld** or if the symbols and relocation have been removed by **strip**(1).

The machine type in the header indicates the type of hardware on which the object code can be executed. Sun-2 code runs on Sun-3 systems, but not vice versa. Program files predating release 3.0 are recognized by a machine type of '**0**'. Sun-4 code may not be run on Sun-2 or Sun-3, nor vice versa.

### Header

The header consists of a **exec** structure. The **exec** structure has the form:

```
struct exec {
        unsigned char   a_dynamic:1;    /* has a __DYNAMIC */
        unsigned char   a_toolversion:7; /* version of toolset used to create this file */
        unsigned char   a_machtype;     /* machine type */
        unsigned short  a_magic;        /* magic number */
        unsigned long   a_text;         /* size of text segment */
        unsigned long   a_data;         /* size of initialized data */
        unsigned long   a_bss;          /* size of uninitialized data */
        unsigned long   a_syms;         /* size of symbol table */
        unsigned long   a_entry;        /* entry point */
        unsigned long   a_trsize;       /* size of text relocation */
        unsigned long   a_drsize;       /* size of data relocation */
};
```

The members of the structure are:

**a_dynamic** **1** if the **a.out** file is dynamically linked or is a shared object, **0** otherwise.

**a_toolversion** The version number of the toolset (**as**, **ld**, etc.) used to create the file.

**a_machtype** One of the following:

| | |
|---|---|
| **0** | pre-3.0 executable image |
| **M_68010** | executable image using only MC68010 instructions that can run on a Sun-2 or Sun-3 |
| **M_68020** | executable image using MC68020 instructions that can run only on a Sun-3 |
| **M_SPARC** | executable image using SPARC instructions that can run only on a Sun-4 |

**a_magic** One of the following:

| | |
|---|---|
| **OMAGIC** | An text executable image which is not to be write-protected, so the data segment is immediately contiguous with the text segment. |

NMAGIC A write-protected text executable image. The data segment begins at the first segment boundary following the text segment, and the text segment is not writable by the program. When the image is started with **execve**(2), the entire text and data segments will be read into memory.

ZMAGIC A page-aligned text executable image. the data segment begins at the first segment boundary following the text segment, and the text segment is not writable by the program. The text and data sizes are both multiples of the page size, and the pages of the file will be brought into the running image as needed, and not pre-loaded as with the other formats. This is the default format produced by **ld**(1).

The macro **N_BADMAG** takes an **exec** structure as an argument; it evaluates to 1 if the **a_magic** field of that structure is invalid, and evaluates to 0 if it is valid.

**a_text** The size of the text segment, in bytes.

**a_data** The size of the initialized portion of the data segment, in bytes.

**a_bss** The size of the "uninitialized" portion of the data segment, in bytes. This portion is actually initialized to zero. The zeroes are not stored in the **a.out** file; the data in this portion of the data segment is zeroed out when it is loaded.

**a_syms** The size of the symbol table, in bytes.

**a_entry** The virtual address of the entry point of the program; when the image is started with **execve**, the first instruction executed in the image is at this address.

**a_trsize** The size of the relocation information for the text segment.

**a_drsize** The size of the relocation information for the data segment.

The macros **N_TXTADDR**, **N_DATADDR**, and **N_BSSADDR** give the memory addresses at which the text, data, and bss segments, respectively, will be loaded.

In the ZMAGIC format, the size of the header is included in the size of the text section; in other formats, it is not.

When an **a.out** file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized data), and a stack. For the ZMAGIC format, the header is loaded with the text segment; for other formats it is not.

Program execution begins at the address given by the value of the **a_entry** field.

The stack starts at the highest possible location in the memory image, and grows downwards. The stack is automatically extended as required. The data segment is extended as requested by **brk**(2) or **sbrk**.

**Text and Data Segments**

The text segment begins at the start of the file for ZMAGIC format, or just after the header for the other formats. The **N_TXTOFF** macro returns this absolute file position when given an **exec** structure as argument. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The **N_DATOFF** macro returns the absolute file position of the beginning of the data segment when given an **exec** structure as argument.

**Relocation**

The relocation information appears after the text and data segments. The **N_TRELOFF** macro returns the absolute file position of the relocation information for the text segment, when given an **exec** structure as argument. The **N_DRELOFF** macro returns the absolute file position of the relocation information for the data segment, when given an **exec** structure as argument. There is no relocation information if **a_trsize+a_drsize==0**.

**Relocation (Sun-2 and Sun-3 Systems)**

If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When

the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is added to the bytes in the file. If a byte involves a reference to a relative location, or relocatable segment, then the value stored in the file is an offset from the associated segment.

If relocation information is present, it amounts to eight bytes per relocatable datum as in the following structure:

```
struct reloc_info_68k {
    long     r_address;          /* address which is relocated */
unsigned int  r_symbolnum:24,    /* local symbol ordinal */
    r_pcrel:1,          /* was relocated pc relative already */
    r_length:2,         /* 0=byte, 1=word, 2=long */
    r_extern:1,         /* does not include value of sym referenced */
    r_baserel:1,        /* linkage table relative */
    r_jmptable:1,       /* pc-relative to jump table */
    r_relative:1,       /* relative relocation */
    :1;
};
```

If **r_extern** is 0, then **r_symbolnum** is actually an **n_type** for the relocation (for instance, **N_TEXT** meaning relative to segment text origin.)

**Relocation (Sun-4 System)**

If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is ignored. Unlike the Sun-2 and Sun-3 system, the offset from the associated symbol is kept with the relocation record. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is added to this offset, and the sum is inserted into the bytes in the text or data segment.

If relocation information is present, it amounts to twelve bytes per relocatable datum as in the following structure:

```
enum reloc_type
{
    RELOC_8,            RELOC_16,           RELOC_32,           /* simplest relocs */
    RELOC_DISP8,        RELOC_DISP16,       RELOC_DISP32,       /* Disp's (pc-rel) */
    RELOC_WDISP30,      RELOC_WDISP22,      /* SR word disp's */
    RELOC_HI22,         RELOC_22,           /* SR 22-bit relocs */
    RELOC_13,           RELOC_LO10,         /* SR 13&10-bit relocs */
    RELOC_SFA_BASE,     RELOC_SFA_OFF13,    /* SR S.F.A. relocs */
    RELOC_BASE10,       RELOC_BASE13,       RELOC_BASE22,       /* base_relative pic */
    RELOC_PC10,         RELOC_PC22,         /* special pc-rel pic*/
    RELOC_JMP_TBL,      /* jmp_tbl_rel in pic */
    RELOC_SEGOFF16,     /* ShLib offset-in-seg */
    RELOC_GLOB_DAT,     RELOC_JMP_SLOT,     RELOC_RELATIVE,     /* rtld relocs */
};

struct reloc_info_sparc   /* used when header.a_machtype == M_SPARC */
{
    unsigned long int   r_address;          /* relocation addr (offset in segment) */
    unsigned int        r_index     :24;    /* segment index or symbol index */
    unsigned int        r_extern    : 1;    /* if F, r_index==SEG#; if T, SYM idx */
    int                 : 2;                /* <unused> */
    enum reloc_type     r_type      : 5;    /* type of relocation to perform */
    long int            r_addend;           /* addend for relocation value */
};
```

If **r_extern** is 0, then **r_symbolnum** is actually a **n_type** for the relocation (for instance, N_TEXT meaning relative to segment text origin.)

**Symbol Table**

The **N_SYMOFF** macro returns the absolute file position of the symbol table when given an **exec** structure as argument. Within this symbol table, distinct symbols point to disjoint areas in the string table (even when two symbols have the same name). The string table immediately follows the symbol table; the **N_STROFF** macro returns the absolute file position of the string table when given an **exec** structure as argument. The first 4 bytes of the string table are not used for string storage, but rather contain the size of the string table. This size *includes* the 4 bytes; thus, the minimum string table size is 4. Layout information as given in the include file for the Sun system is shown below.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
struct nlist {
        union {
                char    *n_name;        /* for use when in-memory */
                long    n_strx;         /* index into file string table */
        } n_un;
        unsigned char   n_type;         /* type flag, that is, N_TEXT etc; see below */
        char    n_other;
        short   n_desc;                 /* see <stab.h> */
        unsigned    n_value;            /* value of this symbol (or adb offset) */
};
#define n_hash  n_desc                  /* used internally by ld */
/*
* Simple values for n_type.
*/
#define N_UNDF  0x0                     /* undefined */
#define N_ABS   0x2                     /* absolute */
#define N_TEXT  0x4                     /* text */
#define N_DATA  0x6                     /* data */
#define N_BSS   0x8                     /* bss */
#define N_COMM  0x12                    /* common (internal to ld) */
#define N_FN    0x1f                    /* file name symbol */
#define N_EXT   01                      /* external bit, or'ed in */
#define N_TYPE  0x1e                    /* mask for all the type bits */

/*
* Other permanent symbol table entries have some of the N_STAB bits set.
* These are given in <stab.h>
*/
#define N_STAB  0xe0                    /* if any of these bits set, don't discard */
```

In the **a.out** file a symbol's **n_un.n_strx** field gives an index into the string table. A **n_strx** value of 0 indicates that no name is associated with a particular symbol table entry. The field **n_un.n_name** can be used to refer to the symbol name only if the program sets this up using **n_strx** and appropriate data from the string table. Because of the union in the **nlist** declaration, it is impossible in C to statically initialize such a structure. If this must be done (as when using **nlist**(3)) the file **<nlist.h>** should be included, rather than **<a.out.h>**; this contains the declaration without the union.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader **ld** as the name of a common region whose size is indicated by the value of the symbol.

FILES

| | |
|---|---|
| **/usr/include/a.out.h** | symbolic link to **/usr/include/machine/a.out.h** |
| **/usr/include/machine** | symbolic link to one of **/usr/include/sun[234...]** |

| | |
|---|---|
| **/usr/include/sun2/a.out.h** | Sun-2 **a.out** header |
| **/usr/include/sun3/a.out.h** | Sun-3 **a.out** header |
| **/usr/include/sun4/a.out.h** | Sun-4 **a.out** header |

**SEE ALSO**

**coff**(5), **adb**(1), **as**(1), **cc**(1V), **dbx**(1), **ld**(1), **nm**(1), **strip**(1), **brk**(2), **nlist**(3)

NAME

acct – execution accounting file

SYNOPSIS

**#include <sys/acct.h>**

DESCRIPTION

The **acct**(2) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

```
/*      @(#)acct.h 2.7 87/03/12 SMI; from UCB 7.1 6/4/86*/

/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

/*
 * Accounting structures;
 * these use a comp_t type which is a 3 bits base 8
 * exponent, 13 bit fraction "floating point" number.
 * Units are 1/AHZ seconds.
 */
typedef u_short comp_t;

struct  acct
{
        char        ac_comm[10];  /* Accounting command name */
        comp_t      ac_utime;     /* Accounting user time */
        comp_t      ac_stime;     /* Accounting system time */
        comp_t      ac_etime;     /* Accounting elapsed time */
        time_t      ac_btime;     /* Beginning time */
        uid_t       ac_uid;       /* Accounting user ID */
        gid_t       ac_gid;       /* Accounting group ID */
        short       ac_mem;       /* average memory usage */
        comp_t      ac_io;        /* number of disk IO blocks */
        dev_t       ac_tty;       /* control typewriter */
        char        ac_flag;      /* Accounting flag */
};

#define AFORK   0001            /* has executed fork, but no exec */
#define ASU     0002            /* used super-user privileges */
#define ACOMPAT 0004            /* used compatibility mode */
#define ACORE   0010            /* dumped core */
#define AXSIG   0020            /* killed by a signal */

/*
 * 1/AHZ is the granularity of the data encoded in the various
 * comp_t fields.  This is not necessarily equal to hz.
 */
#define AHZ 64

#ifdef KERNEL
#ifdef SYSACCT
```

```
struct  acct        acctbuf;
struct  vnode       *acctp;
#else
#define acct()
#endif
#endif
```

If the process does an **execve**(2), the first 10 characters of the filename appear in *ac_comm*. The accounting flag contains bits indicating whether **execve**(2) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

**acct**(2), **execve**(2), **sa**(8)

## NAME

aliases, addresses, forward – addresses and aliases for sendmail(8)

## SYNOPSIS

**/etc/passwd**
**/etc/aliases**
**/etc/aliases.dir**
**/etc/aliases.pag**
**~/.forward**

## DESCRIPTION

These files contain mail addresses or aliases, recognized by **sendmail**(8), for the local host:

| | |
|---|---|
| **/etc/passwd** | Mail addresses (usernames) of local users. |
| **/etc/aliases** | Aliases for the local host, in ASCII format. This file can be edited to add, update, or delete local mail aliases. |
| **/etc/aliases.{dir,pag}** | The aliasing information from **/etc/aliases**, in binary, **dbm**(3X) format for use by **sendmail**(8). The program **newaliases**(8), which is invoked automatically by **sendmail**(8), maintains these files. |
| **~/.forward** | Addresses to which a user's mail is forwarded (see **Automatic Forwarding**, below). |

In addition, the Yellow Pages aliases map *mail.aliases* contains addresses and aliases available for use across the network.

## ADDRESSES

As distributed, **sendmail**(8) supports the following types of addresses:

- Local usernames. These are listed in the local host's **/etc/passwd** file.
- Local filenames. When mailed to an absolute pathname, a message can be appended to a file.
- Commands. If the first character of the address is a vertical bar, (|), **sendmail**(8) pipes the message to the standard input of the command the bar precedes.
- DARPA-standard mail addresses of the form:

  *name*@*domain*

  If *domain* does not contain any '.' (dots), then it is interpreted as the name of a host in the current domain. Otherwise, the message is passed to a *mailhost* that determines how to get to the specified domain. Domains are divided into subdomains separated by dots, with the top-level domain on the right. Top-level domains include:

  | | |
  |---|---|
  | .COM | Commerical organizations. |
  | .EDU | Educational organizations. |
  | .GOV | Government organizations. |
  | .MIL | Military organizations. |

  For example, the full address of John Smith could be:

  js@jsmachine.Podunk-U.EDU

  if he uses the machine named "jsmachine" at Podunk University.

- **uucp**(1C) addresses of the form:

  ... [*host*!]*host*!*username*

  These are sometimes mistakenly referred to as "Usenet" addresses. **uucp**(1C) provides links to numerous sites throughout the world for the remote copying of files.

Other site-specific forms of addressing can be added by customizing the **sendmail** configuration file. See the **sendmail**(8), and *System and Network Administration* for details. Standard addresses are recommended.

## ALIASES

### Local Aliases

**/etc/aliases** is formatted as a series of lines of the form

*name*: *address*[, *address*]

*name* is the name of the alias or alias group, and *address* is the address of a recipient in the group. Aliases can be nested. That is, an *address* can be the name of another alias group. Because of the way **sendmail** performs mapping from upper-case to lower-case, an *address* that is the name of another alias group must not contain any upper-case letters.

Lines beginning with white space are treated as continuation lines for the preceding alias. Lines beginning with # are comments.

### Special Aliases

An alias of the form:

**owner**–*aliasname*: *address*

directs error-messages resulting from mail to *alias-name* to *address*, instead of back to the person who sent the message.

An alias of the form:

*aliasname*: **:include:***pathname*

with colons as shown, adds the recipients listed in the file *pathname* to the *aliasname* alias. This allows a private list to be maintained separately from the aliases file.

### YP Domain Aliases

Normally, the aliases file on the master YP server is used for the *mail.aliases* YP map, which can be made available to every YP client. Thus, the **/etc/aliases*** files on the various hosts in a network will one day be obsolete. Domain-wide aliases should ultimately be resolved into usernames on specific hosts. For example, if the following were in the domain-wide alias file:

**jsmith:js@jsmachine**

then any YP client could just mail to **jsmith** and not have to remember the machine and user name for John Smith. If a .SM YP alias does not resolve to an address with a specific host, then the name of the YP domain is used. There should be an alias of the domain name for a host in this case. For example, the alias:

**jsmith:root**

sends mail on a YP client to **root@podunk-u** if the name of the YP domain is **podunk-u**.

### Automatic Forwarding

When an alias (or address) is resolved to the name of a user on the local host, **sendmail** checks for a **.forward** file, owned by the intended recipient, in that user's home directory, and with universal read access. This file can contain one or more addresses or aliases as described above, each of which is sent a copy of the user's mail.

Care must be taken to avoid creating addressing loops in the **.forward** file. When forwarding mail between machines, be sure that the destination machine does not return the mail to the sender through the operation of any YP aliases. Otherwise, copies of the message may "bounce." Usually, the solution is to change the YP alias to direct mail to the proper destination.

A backslash before a username inhibits further aliasing. For instance, to invoke the **vacation**(1) program, user **js** creates a **.forward** file that contains the line:

**\js, "|/usr/ucb/vacation js"**

so that one copy of the message is sent to the user, and another is piped into the **vacation**(1) program.

**FILES**

**/etc/passwd**
**/etc/aliases**
**~/.forward**

**SEE ALSO**

**uucp**(1C), **vacation**(1), **dbm**(3X), **newaliases**(8), **sendmail**(8),

*System and Network Administration*

**BUGS**

Because of restrictions in **dbm**(3X) a single alias cannot contain more than about 1000 characters. Nested aliases can be used to circumvent this limit.

NAME

ar – archive (library) file format

SYNOPSIS

**#include <ar.h>**

DESCRIPTION

The archive command **ar** combines several files into one. Archives are used mainly as libraries to be searched by the link-editor **ld**(1).

A file produced by **ar** has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG   "!<arch>\n"
#define SARMAG 8

#define ARFMAG "`\n"

struct ar_hdr {
        char    ar_name[16];
        char    ar_date[12];
        char    ar_uid[6];
        char    ar_gid[6];
        char    ar_mode[8];
        char    ar_size[10];
        char    ar_fmag[2];
};
```

The name is a blank-padded string. The **ar_fmag** field contains ARFMAG to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for **ar_mode**, which is octal. The date is the modification date of the file at the time of its insertion into the archive.

Each file begins on a even (0 mod 2) boundary; a NEWLINE is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

Sun386i DESCRIPTION

The file produced by *ar* on Sun386i systems is identical to that described above with the following changes:

Each archive containing COFF files [see **coff**(5)] includes an archive symbol table. This symbol table is used by the link editor **ld** to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by **ar**.

The *ar_name* field of the ar_hdr structure described above is blank-padded and slash (/) terminated. Common format archives can be moved from system to system as long as the portable archive command **ar** is used. Conversion tools such as **convert** exist to aid in the transportation of non-common format archives to this format.

Each archive file member begins on an even byte boundary; a NEWLINE is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

If the archive symbol table exists, the first file in the archive has a zero length name (i.e., **ar_name[0]** == '/' ). The contents of this file are as follows:

- The number of symbols. Length: 4 bytes.
- The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".

- The name string table. Length: *ar_size* – (4 bytes * ("the number of symbols" + 1)).

The number of symbols and the array of offsets are managed with *sgetl* and *sputl*. The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

**SEE ALSO**

**ar**(1V), **ld**(1), **nm**(1)

**Sun386i WARNINGS**

*strip*(1) will remove all archive symbol entries from the header. The archive symbol entries must be restored via the ts option of the *ar*(1V) command before the archive can be used with the link editor *ld*(1).

**BUGS**

Filenames lose trailing blanks. Most software dealing with archives takes even an included blank as a name terminator.

NAME

audit.log – the security audit trail file

SYNOPSIS

```
#include <sys/label.h>
#include <sys/audit.h>
#include <sys/user.h>
```

DESCRIPTION

The **audit.log** file begins with a header record consisting of an **audit_header** structure followed by the previous audit file name. When the audit daemon is started (usually only at boot time), the previous audit file name is NULL.

```
struct audit_header {
        int     ah_magic;       /* magic number */
        time_t  ah_time;        /* the time */
        short   ah_namelen;     /* length of file name */
};
typedef struct audit_header audit_header_t;
```

The file may end with a trailer record consisting of an **audit_trailer** structure followed by the name of the next audit file.

```
struct audit_trailer {
        short   at_record_size;         /* size of this */
        short   at_record_type;         /* its type, a trailer */
        time_t  at_time;                /* the time */
        short   at_namelen;             /* length of file name */
};
typedef struct audit_trailer audit_trailer_t;
```

The **audit.log** file contains audit records in their raw form. The records are of varying size depending on the record type. Each record has a header which is an **audit_record** structure.

```
struct audit_record {
        short           au_record_size;         /* size of this */
        short           au_record_type;         /* its type */
        time_t          au_time;                /* the time */
        short           au_uid;                 /* real uid */
        short           au_auid;                /* audit uid */
        short           au_euid;                /* effective */
        short           au_gid;                 /* real group */
        short           au_pid;                 /* effective */
        int             au_errno;               /* error code */
        int             au_return;              /* a return value */
        blabel_t        au_label;               /* also ... */
        short           au_param_count;                 /* # of parameters */
};
typedef struct audit_record audit_record_t;
```

Immediately following the header is a set of two byte integers, the number of which exist for a given record is contained in the **au_param_count** field. These numbers are the lengths of the additional data items. The additional data items follow the list of lengths, the first length describing the first data item. Interpretation of this data is left to the program accessing it.

**SEE ALSO**

**audit**(2), **getauditfile**(2), **getuseraudit**(2), **audit**(8),

*Security Features Guide*

**NAME**

audit_control – control information for system audit daemon

**SYNOPSIS**

**/etc/security/audit/audit_control**

**DESCRIPTION**

The **audit_control** file contains audit control information read by **auditd**(8). Each line consists of a title and a string, separated by a colon. There are no restrictions on the order of lines in the file, although some lines must appear only once. A line beginning with '#' is a comment.

Directory definition lines list the directories to be used when creating audit files, in the order in which they are to be used. The format of a directory line is:

**dir:** *directory-name*

where *directory-name* is the name of a directory in which to create audit files, with the form:

**/etc/security/audit/***server***/***machine*

where *server* is the name of an audit file system on the machine where this audit directory resides, and *machine* is the name of the local machine, since audit files belonging to different machines are, by convention, stored in separate subdirectories of a single audit directory. The naming convention normally has *server* be the name of a server machine, and all clients mount **/etc/security/audit/***server* at the same location in their local file systems. If the same server exports several different file systems for auditing, their *server* names will, of course, be different.

The audit threshold line specifies the percentage of free space that must be present in the file system containing the current audit file. The format of the threshold line is:

**minfree:** *percentage*

where *percentage* is indicates the amount of free space required. If free space falls below this threshold, the audit daemon **auditd**(8) invokes the shell script **/etc/security/audit/audit_warn.** If no threshold is specified, the default is 0%.

The audit flags line specifies the default system audit value. This value is combined with the user audit value read from **/etc/security/passwd.adjunct** to form the process audit state. The user audit value overrides the system audit value. The format of a flags line is:

**flags:** *audit-flags*

where *audit-flags* specifies which event classes are to be audited. The character string representation of *audit-flags* contains a series of flag names, each one identifying a single audit class, separated by commas. A name preceded by – means that the class should be audited for failure only; successful attempts are not audited. A name preceded by + means that the class should be audited for success only; failing attempts are not audited. Without a prefix, the name indicates that the class is to be audited for both successes and failures. The special string **all** indicates that all events should be audited; **–all** indicates that all failed attempts are to be audited, and **+all** all successful attempts. The prefixes ^, ^–, and ^+ turn off flags specified earlier in the string (^– and ^+ for failing and successful attempts, ^ for both). They are typically used to reset flags.

The following table lists the audit classes:

| short name | long name | short description |
|---|---|---|
| dr | data_read | Read of data, open for reading, etc. |
| dw | data_write | Write or modification of data |
| dc | data_create | Creation or deletion of any object |
| da | data_access_change | Change in object access (modes, owner) |
| lo | login_logout | Login, logout, creation by **at**(1) |
| ad | administrative | Normal administrative operation |
| p0 | minor_privilege | Privileged operation |
| p1 | major_privilege | Unusual privileged operation |

**EXAMPLE**

Here is a sample **/etc/security/audit_control** file for the machine eggplant:

```
dir: /etc/security/audit/jedgar/eggplant
dir: /etc/security/audit/jedgar.aux/eggplant
#
# Last-ditch audit file system when jedgar fills up.
#
dir: /etc/security/audit/global/eggplant
minfree: 20
flags: lo,p0,p1,ad,-all,^-da
```

This identifies server **jedgar** with two file systems normally used for audit data, another server **global** used only when **jedgar** fills up or breaks, and specifies that the warning script is run when the file systems are 80% filled. It also specifies that all logins, privileged and administrative operations are to be audited (whether or not they succeed), and that failures of all types except failures to access data are to be audited.

**FILES**

**/etc/security/audit/audit_control**
**/etc/security/audit/audit_warn**
**/etc/security/audit/*/*/***
**/etc/security/passwd_adjunct**

**SEE ALSO**

**at**(1), **audit**(2), **getfaudflgs**(3), **audit.log**(5), **audit**(8), **auditd**(8)

NAME

audit_data – current information on audit daemon

SYNOPSIS

**/etc/security/audit/audit_data**

DESCRIPTION

The **audit_data** file contains information about the audit daemon. The file contains the process ID of the audit daemon, and the pathname of the current audit log file. The format of the file is:

*<pid>:<pathname>*

Where *pid* is the process ID for the audit daemon, and *pathname* is the full pathname for the current audit log file.

EXAMPLE

**64:/etc/security/audit/auditserv/auditclient/2df0504**

FILES

**/etc/security/audit/audit_data**

SEE ALSO

**audit**(2), **audit.log**(5), **audit**(8), **auditd**(8)

**NAME**

auto.home– automount map for home directories

**SYNOPSIS**

**/etc/auto.home**

**AVAILABILTITY**

Sun386i systems only.

**DESCRIPTION**

**auto.home** resides in the **/etc** directory, and contains **automount**(8) map entries for user's home directories. On Sun386i systems, this file is used to build the **auto.home** Yellow Pages map used by **automount** at system startup and reads the **auto.master** Yellow Pages database, which contains an entry for **auto.home** and **/home .** The **auto.home** map contains entries for each username in the YP **passwd** map, and the *hostname*:*/directory* to NFS mount.

References to **/home/***username* are translated by the automount daemon using the **auto.home** map, and cause the directory specified in the map entry to be nfs mounted and that directory returned to the user's program.

User accounts created using **snap**(1) or **logintool**(8) have **passwd**(5) entries where the initial (home) directory name is, in the form **/home/***username*. **snap** and **logintool** also automatically create the **auto.home** entry for a user account. The format of the entry is described in **automount**(8). An example entry is:

**mtravis system2:/export/home/users/mtravis**

Thus when the user **mtravis** logs into a Sun386i systems, the automounter automatically mounts his home directory from **system2**. This allows a user to log in to any RR workstation on the network and be automatically placed in his or her home directory.

The convention for the format of home directory names used by **snap** and **logintool** is:

**/export/home/***groupname/username*

Note that this is a different map and mechanism for home directories than the one that the automount daemon provides with the **–homes** switch. This is because the Sun386i convention for the format of home directory names differs and provides directories that can be used as mount points on a per user and per group basis.

**FILES**

**/etc/auto.home**

**SEE ALSO**

**snap**(1), **passwd**(5), **automount**(8), **logintool**(8)

**NAME**

auto.vol– automount map for volumes

**SYNOPSIS**

**/etc/auto.vol**

**AVAILABILTITY**

Sun386i systems only.

**DESCRIPTION**

**auto.vol** resides in the **/etc** directory, and contains automount(8) map entries for **volumes**. On Sun386i systems, this file is used to build the **auto.vol** Yellow Pages map used by **automount**(8) at system startup. **automouunt** reads the **auto.master** Yellow Pages map, which contains an entry for **auto.vol** and **/vol.**

References to **/vol/***volume_name* are translated by the automount daemon using the **auto.vol** map, and cause the directory specified in the map entry to be mounted.

The concept of a volume is that it is a self contained directory hierarchy that can be NFS mounted. It is referenced using a known *volume_name*. The use of an automount map is suggested so that the volume and its contents can be referenced through **/vol**. This is advantageous because location-transparency (i.e., which host the volume is on) and replication of read-only volumes can be provided using the automount mechanism. The format of the entry is described in **automount** An example entry is:

**archive system4:/export/archive**

In the above example, the **archive** volume is currently on line on **system4**. Users and programs can reference it via **/vol/archive**. If for some reason the volume had to be moved to another system, **system2** for example, the network or system administrator simply edits the map entry for the archive volume and changes the hostname to **system2** and then rebuilds the Yellow Pages maps.

**archive system2:/export/archive**

Users and programs can continue to refer to the archive volume using **/vol/archive,** unaware that the volume was moved to another system.

**FILES**

**/etc/auto.vol**

**SEE ALSO**

**automount**(8)

**NAME**

bar – tape archive file format

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

**bar**(1), (the tape archive command) dumps several files into one, in a medium suitable for transportation. This format is not compatible with the format generated by **tar**(1).

A "bar tape" or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block that describes the file, followed by zero or more blocks that give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an end-of-file indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the **bar**(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads, unless the **B** keyletter is used.

The header block looks like:

```
#define TBLOCK      512

union hblock {
        char dummy[TBLOCK];
        struct header {
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char rdev[8];
                char linkflag;
                char bar_magic[2];
                char volume_num[4];
                char compressed;
                char date[12];
                char start_of_name;
        } dbuf;
};
```

*start_of_name* is a null-terminated string. *date* is the date of the archive. *bar_magic* is a special number indicating that this is a **bar** archive. *rdev* is the device type, for files that are devices. The other fields are zero-filled octal numbers in ASCII. Each field (of width w) contains w-2 digits, a space, and a null, except *size*, *rdev*, and *mtime*, which do not contain the trailing null. *start_of_name* is the name of the file, as specified on the *bar* command line. Files dumped because they were in a directory that was named in the command line have the directory name as prefix and */filename* as suffix. *mode* is the file mode, with the top bit masked off. *uid* and *gid* are the user and group numbers that own the file. *size* is the size of the file in bytes. Links and symbolic links, and special files, are dumped with this field specified as zero. *mtime* is the modification time of the file at the time it was dumped. *chksum* is a decimal ASCII value that represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *linkflag* is ASCII 0 if the file is "normal" or a special file, 1 if it is an hard link, 2 if it is a symbolic link, and 3 if it is a special file (device or FIFO). The name linked-to, if any, is in a null-terminated string, following *start_of_name*. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

When the **H** modifier is used with **bar** , an additional header block (one that does not pertain to a particular file) is written to the first block of each volume of the archive. The header ID, as specified on the command line, is copied to *start_of_name*. The size reflects the number of bytes to skip to the start of the first full file (always zero on the first volume).

The encoding of the header is designed to be portable across machines.

SEE ALSO

**bar**(1)

**NAME**

bootparams – boot parameter data base

**SYNOPSIS**

**/etc/bootparams**

**DESCRIPTION**

The **bootparams** file contains the list of client entries that diskless clients use for booting. For each diskless client the entry should contain the following information:

name of client
a list of keys, names of servers, and pathnames.

The first item of each entry is the name of the diskless client. The subsequent item is a list of keys, names of servers, and pathnames.

Items are separated by TAB characters.

**EXAMPLE**

Here is an example of the **/etc/bootparams** taken from a SunOS system.

```
myclient        root=myserver:/nfsroot/myclient \
        swap=myserver:/nfsswap/myclient \
        dump=myserver:/nfsdump/myclient
```

**FILES**

**/etc/bootparams**

**SEE ALSO**

**bootparamd**(8)

NAME

COFF – common assembler and link editor output

SYNOPSIS

**#include <filehdr.h>**
**#include <aouthdr.h>**
**#include <scnhdr.h>**
**#include <reloc.h>**
**#include <linenum.h>**
**#include <storclass.h>**
**#include <syms.h>**

AVAILABILITY

Sun386i systems only.

DESCRIPTION

The output from the link editor and the assembler (named **a.out** by default) is in COFF format (Common Object File Format) on the Sun386i system.

A common object file consists of a file header, a system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
UNIX system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **–s** option of **ld**(1) or if they were removed by **strip**(1). Also note that the relocation information will be absent after linking unless the **–r** option of **ld**(1) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment starts at location 0x1000 by default.

The **a.out** file produced by **ld**(1) has the magic number 0413 in the first field of the system header. The headers (file header, system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address will equal 0x1000 plus the size of the headers, and will vary depending upon the number of section headers in the **a.out** file. In an **a.out** file with three sections (.text, .data, .bss, and .comment), the first text address is at 0x000010D0. The text segment is not writable by the program; if other processes are executing the same **a.out** file, the processes will share a single text segment.

The data segment starts at the next 4K boundary past the last text address. The first data address is determined by the following: If an **a.out** file were split into 4K chunks, one of the chunks would contain both the end of text and the beginning of data. When the a.out file is loaded into memory for execution, that chunk will appear twice; once at the end of text and once at the beginning of data (with some unused space in between). The duplicated chunk of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the page size without having to realign the beginning of the data section to a page boundary. Therefore the first data address is the sum of the next segment boundary past the end of text plus the remainder of the last text address divided by 4K. If the last text address is a multiple of 4K no duplication is necessary.

On the Sun386i computer the stack begins at location 0xFBFFFFFF and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the **brk**(2) system call.

For relocatable files the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an "external symbol", and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

**File Header**

The format of the file header is:

```
struct filehdr
{
        unsigned short  f_magic;  /* magic number */
        unsigned short  f_nscns;  /* number of sections */
        long            f_timdat; /* time and date stamp */
        long            f_symptr; /* file ptr to symtab */
        long            f_nsyms;  /* # symtab entries */
        unsigned short  f_opthdr; /* sizeof(opt hdr) */
        unsigned short  f_flags;  /* flags */
};
```

**SunOS System Header**

The format of the system header is:

```
typedef struct aouthdr
{
        short   magic;          /* magic number */
        short   vstamp;         /* version stamp */
        long    tsize;          /* text size in bytes, padded */
        long    dsize;          /* initialized data (.data) */
        long    bsize;          /* uninitialized data (.bss) */
        long    entry;          /* entry point */
        long    text_start;     /* base of text used for this file */
        long    data_start;     /* base of data used for this file */
} AOUTHDR;
```

**Section Header**

The format of the section header is:

```
struct scnhdr
{
        char            s_name[SYMNMLEN];/* section name */
        long            s_paddr;  /* physical address */
        long            s_vaddr;  /* virtual address */
        long            s_size;   /* section size */
        long            s_scnptr; /* file ptr to raw data */
        long            s_relptr; /* file ptr to relocation */
        long            s_lnnoptr;/* file ptr to line numbers */
        unsigned shorts_nreloc;  /* # reloc entries */
        unsigned shorts_nlnno;   /* # line number entries */
        long            s_flags;  /* flags */
};
```

**Relocation**

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```
struct reloc
{
        long        r_vaddr;  /* (virtual) address of reference */
        long        r_symndx; /* index into symbol table */
        ushort      r_type;   /* relocation type */
};
```

The start of the relocation information is *s_relptr* from the section header. If there is no relocation information, *s_relptr* is 0.

**Line Number**

The cc(1V) command generates an entry in the object file for each C source line on which a breakpoint is possible (when invoked with the –g option. Users can refer to line numbers when using the appropriate debugger, such as dbx(1)). The structure of these line number entries appears below.

```
struct lineno
{
        union
        {
                long    l_symndx ;
                long    l_paddr ;
        }               l_addr ;
        unsigned shortl_lnno ;
} ;
```

Numbering starts with one at the top of the source file and increments independent of transition between functions. The initial line number entry for a function has *l_lnno* equal to zero, and the symbol table index of the function's entry is in **l_symndx**. Otherwise, *l_lnno* is non-zero, and *l_paddr* is the physical address of the code for the referenced line. Thus the overall structure is the following:

| *l_addr* | *l_lnno* |
|---|---|
| function symtab index | 0 |
| physical address | line |
| physical address | line |
| ... | |

```
function symtab index   0
physical address        line
physical address        line
...
```

**Symbol Table**

The format of each symbol in the symbol table is described by the syment structure, shown below. This structure is compatible with System V COFF, but has an added _n_dbx structure which is needed by **dbx** (1).

```
#define SYMNMLEN 8
#define FILNMLEN 14
#define DIMNUM   4

struct syment
{
  union                         /* all ways to get a symbol name */
  {
    char        _n_name[SYMNMLEN]; /* name of symbol */
    struct
    {
      long      _n_zeroes;      /* == 0L if in string table */
      long      _n_offset;      /* location in string table */
    } _n_n;
    char        *_n_nptr[2];    /* allows overlaying */
    struct
    {
      char      _n_leading_zero; null char */
      char      _n_dbx_type;  /* stab type */
      short     _n_dbx_desc;  /* value of desc field */
      long      _n_stab_ptr;  /* table ptr */
    } _n_dbx;
  } _n;
  long          n_value;      /* value of symbol */
  short         n_scnum;      /* section number */
  unsigned short n_type;      /* type and derived type */
  char          n_sclass;     /* storage class */
  char          n_numaux;     /* number of aux entries */
};

#define n_name      _n._n_name
#define n_zeroes    _n._n_n._n_zeroes
#define n_offset    _n._n_n._n_offset
#define n_nptr      _n._n_nptr[1]
```

The storage class member (*n_sclass*) is set to one of the constants defined in **<storclass.h>**. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent {
        struct {
                long    x_tagndx;
                union {
                        struct {
                                unsigned short  x_lnno;
                                unsigned short  x_size;
                        } x_lnsz;
                        long    x_fsize;
                } x_misc;
                union {
                        struct {
                                long    x_lnnoptr;
                                long    x_endndx;
                        } x_fcn;
                        struct {
                                unsigned short  x_dimen[DIMNUM];
                        } x_ary;
                } x_fcnary;
                unsigned short  x_tvndx;
        } x_sym;

        struct {
                char    x_fname[FILNMLEN];
        } x_file;

        struct {
                long        x_scnlen;
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        } x_scn;

        struct {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        } x_tv;
};
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is *f_symptr* (from the file header) bytes from the beginning of the file. If the symbol table is stripped, *f_symptr* is 0. The string table (if one exists) begins at *f_symptr* + (*f_nsyms* * SYMESZ) bytes from the beginning of the file.

SEE ALSO

as(1), cc(1V), ld(1), brk(2), ldfcn(3)

## NAME

core – format of memory image file

## SYNOPSIS

**#include <sys/core.h>**

## DESCRIPTION

The operating system writes out a memory image of a terminated process when any of various errors occur. See **sigvec**(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The memory image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). Set-user-ID and set-group-ID programs do not produce core files when they terminate as this would cause a security loophole.

The maximum size of a **core** file is limited by **setrlimit** (see **getrlimit**(2)). Files which would be larger than the limit are not created.

The core file consists of a **core** structure, as defined in the **<sys/core.h>** file, followed by the data pages and then the stack pages of the process image. The **core** structure includes the program's header, the size of the text, data, and stack segments, the name of the program and the number of the signal that terminated the process. The program's header is described by the **exec** structure defined in the **<sys/exec.h>** file, except on Sun386i systems.

```
struct core {
        int     c_magic;        /* Corefile magic number */
        int     c_len;          /* Sizeof (struct core) */
        struct  regs c_regs;    /* General purpose registers */
        struct  exec c_aouthdr; /* A.out header */
        int     c_signo;        /* Killing signal, if any */
        int     c_tsize;        /* Text size (bytes) */
        int     c_dsize;        /* Data size (bytes) */
        int     c_ssize;        /* Stack size (bytes) */
        char    c_cmdname[CORE_NAMELEN + 1]; /* Command name */
        struct  fpu c_fpu;      /* external FPU state */
        int     c_ucode;        /* Exception no. from u_code */
};
```

The members of the structure are:

**c_magic** The magic number **CORE_MAGIC** , as defined in **<sys/core.h>**.

**c_len** The length of the **core** structure in the core file. This need not be equal to the current size of a **core** structure as defined in **<sys/core.h>**, as the core file may have been produced on a different release of the SunOS operating system.

**c_regs** The general purpose registers at the time the core file was produced. This structure is machine-dependent.

**c_aouthdr** The executable image header of the program.

**c_signo** The number of the signal that terminated the process; see **sigvec**(2).

**c_tsize** The size of the text segment of the process at the time the core file was produced.

**c_dsize** The size of the data segment of the process at the time the core file was produced. This gives the amount of data space image in the core file.

**c_ssize** The size of the stack segment of the process at the time the core file was produced. This gives the amount of stack space image in the core file.

**c_cmdname** The first **CORE_NAMELEN** characters of the last component of the path name of the program.

**c_fpu** The status of the floating point hardware at the time the core file was produced. This member is not present on Sun-2s.

**c_ucode** The signal code of the signal that terminated the process, if any. See **sigvec**(2).

SEE ALSO

**adb**(1), **dbx**(1), **getrlimit**(2), **sigvec**(2)

**NAME**

cpio – format of cpio archive

**DESCRIPTION**

The old format *header* structure, when the **–c** option of **cpio** is not used, is:

```
struct {
        short   h_magic,
                h_dev;
        ushort  h_ino,
                h_mode,
                h_uid,
                h_gid;
        short   h_nlink,
                h_rdev,
                h_mtime[2],
                h_namesize,
                h_filesize[2];
        char    h_name[h_namesize rounded to a word];
} Hdr;
```

The byte order here is that of the machine on which the tape was written. If the tape is being read on a machine with a different byte order, you have to use **swab**(3) after reading the header. You can determine what byte order the tape was written with by examining the *h_magic* field; if it is equal to 0143561 (octal), which is the standard magic number 070707 (octal) with the bytes swapped, the tape was written in a byte order opposite to that of the machine on which it is being read. If you are producing a tape to be read on a machine with the opposite byte order to that of the machine on which it is being produced, you can use **swap** before writing the header.

When the –c option is used, the *header* information is described by the statement below:

```
sscanf(Chdr, "%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
        &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
        &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
        &Hdr.h_mtime, &Hdr.h_namesize, &Hdr.h_filesize, &Hdr.h_name);
```

*Longtime* and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The contents of each file is recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in **stat**(2). The length of the NULL-terminated path name *h_name*, including the NULL byte, is given by *h_namesize*.

The last record of the *archive* always contains the name **TRAILER!!!**. Special files, directories, and the trailer, are recorded with *h_filesize* equal to zero. Symbolic links are recorded similarly to regular files, with the "contents" of the file being the name of the file the symbolic link points to.

**SEE ALSO**

**cpio**(1), **find**(1), **stat**(2), **swab**(3)

NAME

crontab – table of times to run periodic jobs

SYNOPSIS

**/var/spool/cron/crontabs/***

DESCRIPTION

The **cron** utility is a permanent process, started by **/etc/rc.local**. **cron** consults the files in the directory **/var/spool/cron/crontabs** to find out what tasks are to be done, and at what time.

Each line in a **crontab** file consists of six fields, separated by spaces or tabs, as follows:

1. Minutes field, which can have values in the range 0 through 59.
2. Hours field, which can have values in the range 0 through 23.
3. Day of the month, in the range 1 through 31.
4. Month of the year, in the range 1 through 12.
5. Day of the week, in the range 0 through 6. Sunday is day 0 in this scheme of things. For backward compatibility with older systems, Sunday may also be specified as day 7.
6. (The remainder of the line) is the command to be run. A percent character in this field (unless escaped by \) is translated to a NEWLINE character. Only the first line (up to a % or end of line) of the command field is executed by the Shell. The other lines are made available to the command as standard input.

Any of fields 1 through 5 can be a list of values separated by commas. A value can either be a number, or a pair of numbers separated by a hyphen, indicating that the job is to be done for all the times in the specified range. If a field is an asterisk character (*) it means that the job is done for all possible values of the field.

Note: the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example,

**0 0 1,15 * 1**

would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example,

**0 0 * * 1**

would run a command only on Mondays).

The command is run from your home directory with an **arg0** of **sh**. Users who desire to have their **.profile** executed must explicitly do so in the command. **cron** supplies a default environment for every shell, defining **HOME, LOGNAME, USER, SHELL(=/bin/sh)**, and **PATH(=:/usr/ucb:/bin:/usr/bin)**.

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

EXAMPLE

```
0 0 * * * calendar –
15 0 * * * /usr/etc/sa –s >/dev/null
15 4 * * * find /etc/preserve –mtime +7 –a –exec rm –f {} ;
40 4 * * * find / –name '#*' –atime +3 –exec rm –f {} ;
0 0 1-5 * * /usr/local/weekdays
0 0 0,6 * * /usr/local/weekends
```

The **calendar** command runs at minute 0 of hour 0 (midnight) of every day. The **/usr/etc/sa** command runs at 15 minutes after midnight every day. The two **find** commands run at 15 minutes past four and at 40 minutes past four, respectively, every day of the year. The **/usr/local/weekdays** command is run at midnight on weekdays. Finally, the **/usr/local/weekends** command is run at midnight on weekends.

**FILES**

**/var/spool/cron/crontabs/*** tables of times to run periodic jobs

**/etc/rc.local**

**.profile**

**SEE ALSO**

**cron**(8), **rc**(8)

NAME

dir – format of directories

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>
```

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory and directories must be read using the **getdirentries**(2) system call or the **directory**(3) library routines. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry; see **fs**(5).

A directory consists of some number of blocks of DIRBLKSIZ bytes, where DIRBLKSIZ is chosen such that it can be transferred to disk in a single atomic operation (512 bytes on most machines):

```
#ifdef KERNEL
#define DIRBLKSIZ DEV_BSIZE
#else
#define DIRBLKSIZ 512
#endif

#define MAXNAMLEN 255
```

Each DIRBLKSIZ byte block contains some number of directory entry structures, which are of variable length. Each directory entry has a **struct direct** at the front of it, containing its inode number, the length of the entry, and the length of the name contained in the entry. These are followed by the name padded to a 4-byte boundary with NULL bytes. All names are guaranteed NULL-terminated. The maximum length of a name in a directory is **MAXNAMLEN**.

The macro **DIRSIZ(dp)** gives the amount of space required to represent a directory entry. Free space in a directory is represented by entries that have:

```
dp->d_reclen > DIRSIZ(dp)
```

All DIRBLKSIZ bytes in a directory block are claimed by the directory entries. This usually results in the last entry in a directory having a large **dp->d_reclen**. When entries are deleted from a directory, the space is returned to the previous entry in the same directory block by increasing its **dp->d_reclen**. If the first entry of a directory block is free, then its **dp->d_ino** is set to 0. Entries other than the first in a directory do not normally have **dp->d_ino** set to 0.

The **DIRSIZ** macro gives the minimum record length which will hold the directory entry. This requires the amount of space in struct direct without the **d_name** field, plus enough space for the name with a terminating NULL byte (**dp->d_namlen+1**), rounded up to a 4-byte boundary.

```
#undef DIRSIZ
#define DIRSIZ(dp)    ((sizeof (struct direct) - (MAXNAMLEN+1)) + (((dp)->d_namlen+1 + 3) &~ 3))
struct   direct {
        u_long  d_ino;
        short   d_reclen;
        short   d_namlen;
        char    d_name[MAXNAMLEN + 1];
        /* typically shorter */
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system ("/"), for which '..' has the same meaning as '.'.

**SEE ALSO**

**getdirentries**(2), **directory**(3), **fs**(5)

## NAME

dump, dumpdates – incremental dump format

## SYNOPSIS

```
#include <sys/types.h>
#include <sys/inode.h>
#include <dumprestor.h>
```

## DESCRIPTION

Tapes used by **dump** and **restore**(8) contain:

a header record
two groups of bit map records
a group of records describing directories
a group of records describing files

The format of the header record and of the first record of each description as given in the include file **<dumprestor.h>** is:

```
#define NTREC  10
#define MLEN            16
#define MSIZ            4096
#define TS_TAPE         1
#define TS_INODE        2
#define TS_BITS 3
#define TS_ADDR         4
#define TS_END  5
#define TS_CLRI6
#define MAGIC   (int) 60011
#define CHECKSUM        (int) 84446
struct  spcl {
        int             c_type;
        time_t          c_date;
        time_t          c_ddate;
        int             c_volume;
        daddr_t         c_tapea;
        ino_t           c_inumber;
        int             c_magic;
        int             c_checksum;
        struct          dinode          c_dinode;
        int             c_count;
        char            c_addr[BSIZE];
} spcl;
struct  idates {
        char            id_name[16];
        char            id_incno;
        time_t          id_ddate;
};
#define  DUMPOUTFMT  "%-16s %c %s"          /* for printf */
                                            /* name, incno, ctime(date) */
#define  DUMPINFMT   "%16s %c %[^\n]\n"     /* inverse for scanf */
```

NTREC is the default number of 1024 byte records in a physical tape block, changeable by the **b** option to **dump**. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are as follows:

| | |
|---|---|
| **TS_TAPE** | Tape volume label |
| **TS_INODE** | A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is. |
| **TS_BITS** | A bit map follows. This bit map has a one bit for each inode that was dumped. |
| **TS_ADDR** | A subrecord of a file description. See *c_addr* below. |
| **TS_END** | End of tape record. |
| **TS_CLRI** | A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped. |
| **MAGIC** | All header records have this number in *c_magic*. |
| **CHECKSUM** | Header records checksum to this value. |

The fields of the header structure are as follows:

| | |
|---|---|
| **c_type** | The type of the header. |
| **c_date** | The date the dump was taken. |
| **c_ddate** | The date the file system was dumped from. |
| **c_volume** | The current volume number of the dump. |
| **c_tapea** | The current number of this (1024-byte) record. |
| **c_inumber** | The number of the inode being dumped if this is of type **TS_INODE**. |
| **c_magic** | This contains the value **MAGIC** above, truncated as needed. |
| **c_checksum** | This contains whatever value is needed to make the record sum to **CHECKSUM**. |
| **c_dinode** | This is a copy of the inode as it appears on the file system; see **fs**(5). |
| **c_count** | The count of characters in *c_addr*. |
| **c_addr** | An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, **TS_ADDR** records will be scattered through the file, each one picking up where the last left off. |

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a **TS_END** record and then the tapemark.

The structure *idates* describes an entry in the file **/etc/dumpdates** where dump history is kept. The fields of the structure are:

| | |
|---|---|
| **id_name** | The dumped filesystem is **/dev/id_nam**. |
| **id_incno** | The level number of the dump tape; see **dump**(8). |
| **id_ddate** | The date of the incremental dump in system format see **types**(5). |

## FILES

**/etc/dumpdates**
**/dev/id_nam**

## SEE ALSO

**fs**(5), **types**(5), **dump**(8), **restore**(8)

## BUGS

Should more explicitly describe format of dumpdates file.

## NAME

environ – user environment

## SYNOPSIS

**extern char **environ;**

## DESCRIPTION

An array of strings called the 'environment' is made available by **execve**(2) when a process begins. By convention these strings have the form '*name=value*'. The following names are used by various commands:

| | |
|---|---|
| **PATH** | The sequence of directory prefixes that **sh**(1), **time**(1V), **nice**(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by ':'. The **login**(1) process sets **PATH=:/usr/ucb:/bin:/usr/bin.** |
| **HOME** | The name of the user's login directory, set by **login**(1) from the password file **/etc/passwd** (see **passwd**(5)). |
| **TERM** | The type of terminal on which the user is logged in. This information is used by commands, such as **nroff**(1) or **plot**(1G), which may exploit special terminal capabilities. See **/etc/termcap** (**termcap**(5)) for a list of terminal types. |
| **SHELL** | The path name of the user's login shell. |
| **TERMCAP** | The string describing the terminal in TERM, or the name of the **termcap** file, see **termcap**(3X), **termcap**(5). |
| **EXINIT** | A startup list of commands read by **ex**(1), **edit**, and **vi**(1). |
| **USER** **LOGNAME** | The user's login name. |
| **TZ** | The name of the time zone that the user is located in. If **TZ** is not present in the environment, the system's default time zone, normally the time zone that the computer is located in, is used. |

Further names may be placed in the environment by the *export* command and '*name=value*' arguments in **sh**(1), or by the **setenv** command if you use **csh**(1). Arguments may also be placed in the environment at the point of an **execve**(2). It is unwise to conflict with certain **sh**(1) variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS.**

## SYSTEM V DESCRIPTION

The description of the variable **TERMCAP** does not apply to programs built in the System V environment.

## FILES

**/etc/passwd**
**etc/termcap**

## SEE ALSO

**csh**(1), **ex**(1), **login**(1), **nice**(1), **nroff**(1), **plot**(1G), **sh**(1), **time**(1V), **vi**(1), **execve**(2), **getenv**(3), **system**(3), **termcap**(3X), **passwd**(5), **termcap**(5)

**NAME**

ethers – Ethernet address to hostname database or YP domain

**DESCRIPTION**

The **ethers** file contains information regarding the known (48 bit) Ethernet addresses of hosts on the Internet. For each host on an Ethernet, a single line should be present with the following information:

Ethernet address
official host name

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment extending to the end of line.

The standard form for Ethernet addresses is "*x:x:x:x:x:x*" where *x* is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than a SPACE, TAB, NEWLINE, or comment character. It is intended that host names in the **ethers** file correspond to the host names in the **hosts**(5) file.

The **ether_line**( ) routine from the Ethernet address manipulation library, **ethers**(3N) may be used to scan lines of the **ethers** file.

**FILES**

**/etc/ethers**

**SEE ALSO**

**ethers**(3N), **hosts**(5)

## NAME

exports, xtab – directories to export to NFS clients

## SYNOPSIS

**/etc/exports**

**/etc/xtab**

## DESCRIPTION

The **/etc/exports** file contains entries for directories that can be exported to NFS clients. This file is read automatically by the **exportfs**(8) command. If you change this file, you must run **exportfs**(8) for the changes to affect the daemon's operation.

Only when this file is present at boot time does the **rc.local** script execute **exportfs**(8) and start the NFS file-system daemon, **nfsd**(8).

The **/etc/xtab** file contains entries for directories that are *currently* exported. This file should only be accessed by programs using **getexportent** (see **exportent**(3)). (Use the **–u** option of **exportfs** to remove entries from this file).

An entry for a directory consists of a line of the following form:

*directory* *–option*[,*option* ]...

*directory* is the pathname of a directory (or file).

*option* is one of

**ro** Export the directory read-only. If not specified, the directory is exported read-write.

**rw=***hostnames*[**:***hostname*]...
Export the directory read-mostly. Read-mostly means read-only to most machines, but read-write to those specified. If not specified, the directory is exported read-write to all.

**anon=***uid*
If a request comes from an unknown user, use *uid* as the effective user ID. Note: root users (uid 0) are always considered "unknown" by the NFS server, unless they are included in the "root" option below. The default value for this option is –2. Setting "anon" to –1 disables anonymous access. Note: by default secure NFS will accept insecure requests as anonymous, and those wishing for extra security can disable this feature by setting "anon" to –1.

**root=***hostnames*[**:***hostname*]...
Give root access only to the root users from a specified *hostname*. The default is for no hosts to be granted root access.

**access=***client*[**:***client*]...
Give mount access to each *client* listed. A *client* can either be a hostname, or a netgroup (see **netgroup**(5)). Each *client* in the list is first checked for in the netgroup database, and then the hosts database. The default value allows any machine to mount the given directory.

**secure** Require clients to use a more secure protocol when accessing the directory.

A '#' (pound-sign) anywhere in the file indicates a comment that extends to the end of the line.

## EXAMPLE

```
/usr          -access=clients              # export to my clients
/usr/local    # export to the world
/usr2         -access=hermes:zip:tutorial  # export to only these machines
/usr/sun      -root=hermes:zip             # give root access only to these
/usr/new      -anon=0                      # give all machines root access
/usr/bin      -ro                          # export read-only to everyone
/usr/stuff    -access=zip,anon=-3,ro       # several options on one line
```

## FILES

**/etc/exports**
**/etc/xtab**
**/etc/hosts**
**/etc/netgroup**
**rc.local**

## SEE ALSO

**exportent**(3), **hosts**(5), **netgroup**(5), **exportfs**(8), **nfsd**(8)

## WARNINGS

You cannot export either a parent directory or a subdirectory of an exported directory that is *within the same filesystem*. It would be illegal, for instance, to export both **/usr** and **/usr/local** if both directories resided on the same disk partition.

NAME

fcntl – file control options

SYNOPSIS

**#include <fcntl.h>**

DESCRIPTION

The **fcntl**(2V) function provides for control over open files. This include file describes *requests* and *arguments* to **fcntl** and **open**(2V) as shown below:

```
/*              @ (#)fcntl.h 1.2 83/12/08 SMI; from UCB 4.2 83/09/25*/
/*
* Flag values accessible to open(2V) and fcntl(2)
*  (The first three can only be set by open)
*/
#define         O_RDONLY                0
#define         O_WRONLY                1
#define         O_RDWR                  2
#define         O_NDELAY                FNDELAY         /* Non-blocking I/O */
#define         O_APPEND                FAPPEND         /* append (writes guaranteed at the end) */
#ifndef         F_DUPFD
/* fcntl(2) requests */
#define         F_DUPFD                 0               /* Duplicate fildes */
#define         F_GETFD                 1               /* Get fildes flags */
#define         F_SETFD                 2               /* Set fildes flags */
#define         F_GETFL                 3               /* Get file flags */
#define         F_SETFL                 4               /* Set file flags */
#define         F_GETOWN                5               /* Get owner */
#define         F_SETOWN                6               /* Set owner */
/* flags for F_GETFL, F_SETFL— copied from <sys/file.h> */
#define         FNDELAY                                 00004/* non-blocking reads */
#define         FAPPEND                                 00010/* append on each write */
#define         FASYNC                                  00100/* signal pgrp when data ready */
#endif
```

SEE ALSO

**fcntl**(2V), **open**(2V)

NAME

fs, inode – format of a 4.2 (ufs) file system volume

SYNOPSIS

**#include <sys/types.h>**
**#include <ufs/fs.h>**
**#include <ufs/inode.h>**

DESCRIPTION

Standard 4.2 (ufs) file system storage volumes have a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 15 contain primary and secondary bootstrapping programs.

The actual file system begins at sector 16 with the *super-block*. The layout of the super block is defined by the include file **<ufs/fs.h>**

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group contains inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of "blocks." File system blocks of at most size **MAXBSIZE** can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be **DEV_BSIZE**, or some multiple of a **DEV_BSIZE** unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the '**blksize(fs, ip, lbn)**' macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 cannot be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it). The *lost+found* directory is given the next available inode when it is initially created by **mkfs**(8).

**fs_minfree** gives the minimum acceptable percentage of file system blocks which may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. This may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of **fs_minfree** is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4, thus the default fragment size is a fourth of the block size.

*Cylinder group related limits*: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. **NRPOS** is the number of rotational positions which are distinguished. With **NRPOS** 8 the resolution of the summary information is 2ms for a typical 3600 rpm drive.

**fs_rotdelay** gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for **fs_rotdelay** is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each **NBPI** bytes of disk space. The inode allocation strategy is extremely conservative.

MAXIPG bounds the number of inodes per cylinder group, and is needed only to keep the structure simpler by having the only a single variable size element (the free bit map).

Note: MAXIPG must be a multiple of INOPB(fs).

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to (**struct cg**) must keep its size within INBSIZE. MAXCPG is limited only to dimension an array in (**struct cg**); it can be made larger as long as that structure's size remains within the bounds dictated by MINBSIZE. Note: super blocks are never more than size SBSIZE.

The path name on which the file system is mounted is maintained in **fs_fsmnt**. MAXMNTLEN defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. It is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from **fs_csaddr** (size **fs_cssize**) in addition to the super block.

Note: sizeof (**struct csum**) must be a power of two in order for the **fs_cs** macro to work.

*Super block for a file system*: MAXBPC bounds the size of the rotational layout tables and is limited by the fact that the super block is of size SBSIZE. The size of these tables is **inversely** proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (**fs_cpc**). The size of the rotational layout tables is derived from the number of bytes remaining in (**struct fs**).

MAXBPG bounds the number of blocks of data per cylinder group, and is limited by the fact that cylinder groups are at most one block. The size of the free block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure (**struct cg**).

*inode*: The inode is the focus of all file activity in the file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is "named" by its device/i-number pair. For further information, see the include file **<ufs/inode.h>**.

SEE ALSO

**mkfs**(8)

**NAME**

fspec – format specification in text files

**DESCRIPTION**

It is sometimes convenient to maintain text files on the operating system with non-standard tab stop settings, (that is, tab stops that are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all TAB characters with the appropriate number of SPACE characters, before they can be processed by operating system commands. A format specification occurring in the first line of a text file specifies how TAB characters are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

**t** *tabs* The **t** parameter specifies the tab stop settings for the file. The value of *tabs* must be one of the following:

1. A list of column numbers separated by commas, indicating tab stops set at the specified columns;
2. A '–' followed immediately by an integer *n*, indicating tab stops set at intervals of *n* columns, that is, at 1+*n*, 1+2*n*, and so on;
3. A '–' followed by the name of a "canned" tab stop specification.

Up to 40 numbers are allowed in a comma-separated list of tab stop settings. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **t1, 10, 20, 30** and **t1, 10, +10, +10** are considered identical.

Standard tab stops are specified by **t–8**, or equivalently, **t1, 9, 17, 25**, etc. This is the tab stop setting that most operating system utilities assume, and is the most likely setting to be found at a terminal. The specification **t–0** specifies no tab stops at all.

The canned tab stops specifications that are recognized are as follows:

**a** 1, 10, 16, 36, 72
Assembler, IBM S/370, first format

**a2** 1, 10, 16, 40, 72
Assembler, IBM S/370, second format

**c** 1, 8, 12, 16, 20, 55
COBOL, normal format

**c2** 1, 6, 10, 14, 49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a TAB reaches column 12. Files using this tab stop setup should include a format specification as follows:

**<:t–c2 m6 s66 d:>**

**c3** 1, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 67
COBOL compact format (columns 1-6 omitted), with more tab stops than **c2**. This is the recommended format for COBOL. The appropriate format specification is:

**<:t–c3 m6 s66 d:>**

**f** 1, 7, 11, 15, 19, 23
FORTRAN

**p** 1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61
PL/I

**s** 1, 10, 55

SNOBOL

**u** 1, 12, 20, 44
UNIVAC 1100 Assembler

**s** *size* The **s** parameter specifies a maximum line size. The value of **size** must be an integer. Size checking is performed after TAB characters have been expanded, but before the margin is prepended.

**m** *margin*
The **m** parameter specifies a number of SPACE characters to be prepended to each line. The value of *margin* must be an integer.

**d** The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

**e** The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t–8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

*** <:t5,10,15 s72:> ***

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

SEE ALSO

**ed**(1), **tabs**(1)

## NAME

fstab, mtab – static filesystem mounting table, mounted filesystems table

## SYNOPSIS

**/etc/fstab**

**/etc/mtab**

## DESCRIPTION

The **/etc/fstab** file contains entries for filesystems and disk partitions to mount using the **mount**(8) command, which is normally invoked by the **rc.boot** script at boot time. This file is used by various utilities that mount, unmount, check the consistency of, dump, and restore file systems. It is also used by the system itself when locating the swap partition.

The **/etc/mtab** file contains entries for filesystems *currently* mounted, and is read by programs using the routines described in **getmntent**(3). **umount** (see **mount**(8)) removes entries from this file.

Each entry consists of a line of the form:

*filesystem directory type options freq pass*

*filesystem* is the pathname of a block-special device, or the name of a remote filesystem in *host*:*pathname* form.

*directory* is the pathname of the directory on which to mount the filesystem.

*type* is the filesystem type, which can be one of:

| | |
|---|---|
| **4.2** | to mount a block-special device |
| **nfs** | to mount an exported NFS filesystem |
| **swap** | to indicate a swap partition |
| **ignore** | to have the **mount** command ignore the current entry (good for noting disk partitions that are not being used) |

*options* contains a comma-separated list (no spaces) of mounting options, some of which can be applied to all types of filesystems, and others which only apply to specific types.

**4.2** options:

**quota | noquota**
disk quotas are enforced or not enforced

**nfs** options:

| | |
|---|---|
| **bg | fg** | If the first attempt fails, retry in the background, or, in the foreground |
| **retry=***n* | The number of times to retry the mount operation. |
| **rsize=***n* | Set the read buffer size to *n* bytes. |
| **wsize=***n* | Set the write buffer size to *n* bytes. |
| **timeo=***n* | Set the NFS timeout to *n* tenths of a second. |
| **retrans=***n* | The number of NFS retransmissions. |
| **port=***n* | The server IP port number. |
| **soft | hard** | Return an error if the server does not respond, or continue the retry request until the server responds. |
| **intr** | Allow keyboard interrupts on hard mounts. |
| **secure** | Use a more secure protocol for NFS transactions. |
| **acregmin=***n* | Hold cached attributes for at least *n* seconds after file modification. |
| **acregmax=***n* | Hold cached attributes for no more than *n* seconds after file modification. |
| **acdirmin=***n* | Hold cached attributes for at least *n* seconds after directory update. |
| **acdirmax=***n* | Hold cached attributes for no more than *n* seconds after directory update. |
| **actimeo=***n* | Set *min* and *max* times for regular files and directories to *n* seconds. |

Common options:

**ro | rw** mount either read-only or read-write

**suid | nosuid**
setuid execution allowed or disallowed

**grpid** Create files with BSD semantics for propagation of the group ID. With this option, files inherit the group ID of the directory in which they are created, regardless of the directory's setgid bit.

**noauto** Do not mount this file system automatically (using **mount –a**).

*freq* is the interval (in days) between dumps.

*pass* is the **fsck**(8) pass in which to check the partition. Filesystems with the same pass number are checked simultaneously. Filesystems with *pass* equal to 0 are not checked.

A pound-sign (#) as the first non-white character indicates a comment line which is ignored by routines that read this file. The order of records in **/etc/fstab** is important because **fsck, mount,** and **umount** process the file sequentially; an entry for a file system must appear *after* the entry for any file system it is to be mounted on top of.

## EXAMPLES

In this example, the **/home/user** directory is hard mounted read-write over the NFS, along with additional swap space in the form of a mounted swap file (see *System and Network Administration* for details on adding swap space):

```
/dev/xy0a / 4.2 rw,noquota 1 1
/dev/xy0b /usr 4.2 rw,noquota 1 1
example:/home/user /home/user nfs rw,hard,fg 0 0
/export/swap/myswap swap swap rw 0 0
```

## FILES

**/etc/fstab**
**/etc/mtab**

## SEE ALSO

**getmntent**(3), **fsck**(8), **mount**(8), **quotacheck**(8), **quotaon**(8),

**NAME**

ftpusers – list of users prohibited by ftp

**SYNOPSIS**

**/usr/etc/ftpusers**

**DESCRIPTION**

**ftpusers** contains a list of users who cannot access this system using the **ftp**(1C) program. **ftpusers** contains one user name per line.

**SEE ALSO**

**ftp**(1C), **ftpd**(8C)

## NAME

gettytab – terminal configuration data base

## SYNOPSIS

**/etc/gettytab**

## DESCRIPTION

**gettytab** is a simplified version of the **termcap**(5) data base used to describe terminal lines. The initial terminal login process **getty**(8) accesses the **gettytab** file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

There is a default terminal class, **default**, that is used to set global defaults for all other classes. (That is, the **default** entry is read, then the entry for the class required is used to override particular settings.)

## CAPABILITIES

Refer to **termcap**(5) for a description of the file layout. The *Default* column below lists defaults obtained if there is no entry in the table obtained, nor one in the special default table.

| *Name* | *Type* | *Default* | *Description* |
|---|---|---|---|
| **ab** | bool | false | read a \r first and guess the baud rate from it |
| **ap** | bool | false | terminal uses 7 bits, any parity |
| **bd** | num | 0 | backspace delay |
| **bk** | str | 0377 | alternate end of line character (input break) |
| **cb** | bool | false | use crt backspace mode |
| **cd** | num | 0 | carriage-return delay |
| **ce** | bool | false | use crt erase algorithm |
| **ck** | bool | false | use crt kill algorithm |
| **cl** | str | NULL | screen clear sequence |
| **co** | bool | false | console - add after login prompt |
| **dx** | bool | false | set DECCTLQ |
| **ds** | str | ^Y | delayed suspend character |
| **ec** | bool | false | leave echo OFF |
| **ep** | bool | false | terminal uses 7 bits, even parity |
| **er** | str | ^? | erase character |
| **et** | str | ^D | end of text (EOF) character |
| **ev** | str | NULL | initial environment |
| **f0** | num | unused | tty mode flags to write messages |
| **f1** | num | unused | tty mode flags to read login name |
| **f2** | num | unused | tty mode flags to leave terminal as |
| **fd** | num | 0 | form-feed (vertical motion) delay |
| **fl** | str | ^O | output flush character |
| **hc** | bool | false | do NOT hangup line on last close |
| **he** | str | NULL | hostname editing string |
| **hn** | str | hostname | hostname |
| **ht** | bool | false | terminal has real tabs |
| **ig** | bool | false | ignore garbage characters in login name |
| **im** | str | NULL | initial (banner) message |
| **in** | str | ^C | interrupt character |
| **is** | num | unused | input speed |
| **kl** | str | ^U | kill character |
| **lc** | bool | false | terminal has lower case |
| **lm** | str | login: | login prompt |
| **ln** | str | ^V | "literal next" character |
| **lo** | str | /usr/bin/login | program to exec when name obtained |
| **nd** | num | 0 | newline (line-feed) delay |
| **nl** | bool | false | terminal has (or might have) a newline character |

| | | | |
|---|---|---|---|
| **nx** | str | default | next table (for auto speed selection) |
| **op** | bool | false | terminal uses 7 bits, odd parity |
| **os** | num | unused | output speed |
| **p8** | bool | false | terminal uses 8 bits, no parity |
| **pc** | str | | pad character |
| **pe** | bool | false | use printer (hard copy) erase algorithm |
| **pf** | num | 0 | delay between first prompt and following flush (seconds) |
| **ps** | bool | false | line connected to a MICOM port selector |
| **qu** | str | ^ | quit character |
| **rp** | str | ^R | line retype character |
| **rw** | bool | false | do NOT use RAW for input, use CBREAK |
| **sp** | num | 0 | line speed (input and output) |
| **su** | str | ^Z | suspend character |
| **tc** | str | none | table continuation |
| **td** | num | 0 | tab delay |
| **to** | num | 0 | timeout (seconds) |
| **tt** | str | NULL | terminal type (for enviroment) |
| **ub** | bool | false | do unbuffered output (of prompts etc) |
| **uc** | bool | false | terminal is known upper case only |
| **we** | str | ^W | word erase character |
| **xc** | bool | false | do NOT echo control chars as ^X |
| **xf** | str | ^S | XOFF (stop output) character |
| **xn** | str | ^Q | XON (start output) character |

If no line speed is specified, speed will not be altered from that which prevails when **getty** is entered. Specifying an input or output speed overrides line speed for stated direction only. If **ab** is specified, **getty** will initially read a character from the tty, assumed to be a carriage return, and will attempt to figure out the baud rate based on what the character appears as. It will then look for a table entry for that baud rate; if the line appears to be a 300 baud line, it will look for an entry **300-baud**, if it appears to be a 1200 baud line, it will look for an entry **1200-baud**, etc..

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the Boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overridden with one of the **f0**, **f1**, or **f2** numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32 bit) value.

Should **getty** receive a NULL character (presumed to indicate a line break) it will restart using the table indicated by the **nx** entry. If there is none, it will re-use its original table.

Delays are specified in milliseconds, the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The **cl** screen clear string may be preceded by a (decimal) number of milliseconds of delay required (as with **termcap**(5)). This delay is simulated by repeated use of the pad character **pc**.

The initial message, and login message, **im** and **lm** may include the character sequence **%h** or **%t** to obtain the hostname or tty name respectively. (**%%** obtains a single '**%**' character.) The hostname is normally obtained from the system, but may be set by the **hn** table entry. In either case it may be edited with **he**. The **he** string is a sequence of characters, each character that is neither '@' nor '#' is copied into the final hostname. A '@' in the **he** string, copies one character from the real hostname to the final hostname. A '#' in the **he** string, skips the next character of the real hostname. Surplus '@' and '#' characters are ignored.

When **getty** execs the login process, given in the **lo** string (usually **/usr/bin/login**), it will have set the enviroment to include the terminal type, as indicated by the **tt** string (if it exists). The **ev** string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form *name=value*.

If a non-zero timeout is specified, with **to**, then **getty** will exit within the indicated number of seconds, either having received a login name and passed control to *login*, or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from **getty** is even parity unless **op** or **p8** is specified. **op** may be specified with **ap** to allow any parity on input, but generate odd parity output. Note: this only applies while **getty** is being run, terminal driver limitations prevent a more complete implementation. **getty** does not check parity of input characters in RAW mode.

**FILES**

**/etc/gettytab**

**SEE ALSO**

**termcap**(5), **getty**(8)

**NAME**

group – group file

**SYNOPSIS**

**/etc/group**

**DESCRIPTION**

The **group** file contains a one-line entry for each group recognized by the system, of the form:

*groupname*:*password*:*gid*:*user-list*

where:

| | |
|---|---|
| *groupname* | is the name of the group. |
| *gid* | is the group's numerical ID within the system; it must be unique. |
| *user-list* | is a comma-separated list of users allowed in the group. |

If the password field is empty, no password is demanded. The **group** file is an ASCII file. Because of the encrypted passwords, the **group** file can and does have general read permission, and can be used as a mapping of numerical group IDs to user names.

A group entry beginning with a '+' (plus sign), means to incorporate an entry or entries from the Yellow Pages. A '+' on a line by itself means to insert the entire contents of the Yellow Pages group file at that point in the file. An entry of the form: '+*groupname*' means to insert the entry (if any) for **groupname**. If a '+' entry has a non-empty *password* or *user-list* field, the contents of that field override the corresponding field from the Yellow Pages. The *gid* field cannot be overridden in this way.

An entry of the form: –*groupname* indicates that the group is disallowed. All subsequent entries for the indicated **groupname**, whether originating from the Yellow Pages, or the local **group** file, are ignored.

Malformed entries cause routines that read this file to halt, in which case group assignments specified further along are never made. To prevent this from happening, use **grpck**(8) to check the **/etc/group** database from time to time.

The Sun386i system uses the following group IDs as program privileges:

| | |
|---|---|
| operator – 5 | Privilege to do backup as root. |
| accounts – 11 | Privilege to update user accounts. |
| networks –12 | Privilege to change network configuration. |
| devices – 13 | Privilege to modify printer, terminal, or modem configurations. |

On all Sun systems, SunOS uses group ID 0 as privilege to run **su**(1).

**EXAMPLE**

Here is a sample group file when the **group.adjunct** file does not exist:

```
primary:q.mJzTnu8icF.:10:fred,mary
+myproject:::bill,steve
+:
```

Here is a sample group file when the **group.adjunct** file does exist:

```
primary:#$primary:10:fred,mary
+myproject:::bill,steve
+:
```

If these entries appear at the end of a group file, then the group *primary* will have members **fred** and **mary**, and a group ID of **10**. The group *myproject* will have members **bill** and **steve**, and the password and group ID of the Yellow Pages entry for the group **myproject**. All groups listed in the Yellow Pages are pulled in and placed after the entry for **myproject**.

**FILES**

**/etc/group**

**SEE ALSO**

**passwd**(1), **su**(1), **getgroups**(2), **initgroups**(3), **crypt**(3), **group.adjunct**(5), **passwd**(5), **grpck**(8)

**BUGS**

The **passwd**(1) command will not change group passwords.

## NAME

group.adjunct – group security data file

## SYNOPSIS

**/etc/security/group.adjunct**

## DESCRIPTION

The **group.adjunct** file contains the following information for each group:

*groupname* This is the group's name in the system; it must be unique.

*password* The encrypted password, formerly field two of the **/etc/group** file.

The **group.adjunct** file is in ASCII. Fields are separated by a colon, and each group is separated from the next by a NEWLINE.

A **group.adjunct** file can have a line beginning with a '+' (plus sign), which means to incorporate entries from the Yellow Pages. There are two styles of '+' entries: all by itself, '+' means to insert the entire contents of the **group.adjunct** Yellow Pages file at that point; *+name* means to insert the entry (if any) for *name* from the Yellow Pages at that point. If a '+' entry has a non-NULL password, the contents of that field will override what is contained in the Yellow Pages.

## FILES

**/etc/group**

## SEE ALSO

**crypt**(3), **getgraent**(3), **getgrent**(3), **group**(5)

**NAME**

help – help file format

**SYNOPSIS**

**/usr/lib/help/***

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

Each SunView application using the **help** feature has a simple ASCII file in **/usr/lib/help** with the name *application-name*.**info**.

This file contains the text of help messages for each SunView object within that program. Each help message is separated in the file by a line beginning with a colon and identified by a keyword which matches the HELP_DATA attribute of the SunView object.

The first character of each line in the file may be:

| | |
|---|---|
| # | comment line |
| : | keyword line |
| any other | 1-32 help text lines |

If the line is a keyword line, it has the following structure—

:keyword[s]:datastring [pagenumber]<cr>

*keyword* is a 1-65 character keyword
--any displayable characters may be used
--several keywords may be present
--keywords are separated by 1-or-more blanks

*datastring* is 1-256 ASCII bytes, and describes the path of the data files for help_viewer, relative to /usr/lib/help.

*pagenumber* is an optional page number within the help_viewer data file.

The help text which follows the :keyword line will be displayed in an Alert Box when help is requested for one of the keywords by pressing the help key.

The datastring will be sent (by RPC) to the **help_viewer** procedure when the user selects the More Help box in the Alert Box window.

**EXAMPLE**

Here is part of a typical help file, called mailtool.info.

```
:abort
  Abort button

o Quits the Mail application (click
  left on button). Tentative message
  deletions do not become permanent.

o Provides a menu of Abort options
  (click right on button).

:cancel:mailtool/Writing_and_Sending_Mail 1
```

**Cancel button**

- **Closes the message composition window without sending message (click left on button).**
- **Provides a menu of Cancel options (click right on button).**

Pressing the help key while in the cancel or abort buttons triggers the display of the corresponding text. The words *cancel* and *abort* in this file are the keywords. In the case of abort, there is no More Help available. For cancel, More Help is available and it is stored in the first page of the Writing_and_Sending_Mail file in the mailtool directory.

**FILES**

**/usr/lib/help/*** files for the pop-up help facility

**SEE ALSO**

**help_viewer**(1), **help_viewer**(5)

*Sun386i Developer's Guide*

**NAME**

help_viewer – help viewer file format

**SYNOPSIS**

**/usr/lib/help/*/***

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

The **help_viewer** reads files of various types. The Top Level list of applications documented is **/usr/lib/help/Top_Level**. The Master Index shown at the top level is **/usr/lib/help/Master_Index**. These files are FrameMaker files. To add or remove a heading from this list, use FrameMaker (1.1 or later).

Each directory within **/usr/lib/help** that corresponds to a SunView application name contains detailed information about that application. These are also FrameMaker files. The ***.rf** files are rasterfiles, of standard image format created by FrameMaker. These are the pictures that are interleaved into the text.

The **Frame/** subdirectory of **/usr/lib/help** contains topic, contents, and index templates which can be used to create new Help Viewer handbooks. The **Interleaf/** subdirectory contains Interleaf templates, fonts, and initialization files.

**FILES**

**/usr/lib/help/*/***

**SEE ALSO**

**help**(5), **help_viewer**(1)

**NAME**

hosts – host name data base

**SYNOPSIS**

**/etc/hosts**

**DESCRIPTION**

The **hosts** file contains information regarding the known hosts on the DARPA Internet. For each host a single line should be present with the following information:

Internet address
official host name
aliases

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional '.' notation using the **inet_addr** ( ) routine from the Internet address manipulation library, **inet**(3N). Host names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

**EXAMPLE**

Here is a typical line from the **/etc/hosts** file:

**192.9.1.20 gaia # John Smith**

**FILES**

**/etc/hosts**

**SEE ALSO**

**gethostent**(3N), **inet**(3N)

## NAME

hosts.equiv, rhosts – trusted hosts by system and by user

## DESCRIPTION

The **/etc/hosts.equiv** file contains a list of trusted hosts. When an **rlogin**(1C) or **rsh**(1C) request is received from a host listed in this file, and when the user making the request is listed in the **/etc/passwd** file, then the remote login is allowed with no further checking. In this case, **rlogin** does not prompt for a password, and commands submitted through **rsh** are executed. Thus, a remote user with a local user ID is said to have "equivalent" access from a remote host named in this file.

The format of the **hosts.equiv** file consists of a one-line entry for each host, of the form:

*hostname* [*username*]

The *hostname* field normally contains the name of a trusted host from which a remote login can be made. However, an entry consisting of a single '+' indicates that all known hosts are to be trusted. A hostname must be the "official" name as listed in the **hosts**(5) database. This is the first name given in the hosts database entry; hostname aliases are not recognized. Remote login access can also be given or denied for all hosts within a specific network group. An entry of the form:

*+@group*

means that all hosts in the named network group are trusted. An entry of the form:

*−@group*

means that all hosts in the group are not trusted; remote login access is denied to hosts in that group, except when an entry for a specific host appears ahead of the "minus" group entry.

The *username* field can be used to specify a user who is allowed to log in under any valid user ID. Careful thought about security should be given before providing this privilege to a user. You can also specify a network group in the *username* field with an entry of the form:

*+@group1 +@group2*

in which case any user in *group2* logging in from a host in *group1* may log in as anyone. Again, security is an important consideration here.

### The User's .rhosts File

Whenever a remote login is attempted, the remote login daemon checks for a **.rhosts** file in the home directory of the user attempting to log in. A user's **.rhosts** file has the same format as the **hosts.equiv** file, and is used to give or deny access only for the *specific user* attempting to log in from a given host. While an entry in the **hosts.equiv** file allows remote login access to *any* user from the indicated host, an entry in a user's **.rhosts** file only allows access from a named host to the user in whose home directory the **.rhosts** file appears. (When this file is used, permissions in the user's home directory should allow read and search access by anyone, so it may be located and read.) When a user attempts a remote login, his **.rhosts** file is, in effect, prepended to the **hosts.equiv** file for permission checking. Thus, if a host is specified in the user's **.rhosts** file, login access is allowed, even if it would otherwise be excluded by a minus group entry in **/etc/hosts.equiv**.

### The Root .rhosts File

When the user attempting a remote login is **root**, only the **/.rhosts** file is checked, not **/etc/hosts.equiv**.

## FILES

**/etc/hosts.equiv**
**/etc/passwd**
**~/.rhosts**
**/etc**

## SEE ALSO

**rlogin**(1C), **rsh**(1C), **hosts**(5), **netgroup**(5), **passwd**(5)

**NAME**

indent.pro – default options for indent

**DESCRIPTION**

The **.indent.pro** file in either the current or home directory contains default command line options for the **indent**(1) program. It is a text file that contains space-seperated command line options. For a description of these options, see **indent**(1).

Explicit command line options override options taken from **.indent.pro.**

Here is a sample **.indent.pro** file:

**–bap –nbad –nbbb –bc –br –cdb –nce**
**–fc1 –ip –lp –npcs –psl –sc –nsob –cli0**
**–di12 –l79 –i4 –d0 –c33**

**FILES**

**./.indent.pro**
**~/.indent.pro**

**SEE ALSO**

**indent**(1)

## NAME

inetd.conf – Internet servers database

## DESCRIPTION

The **inetd.conf** file contains the list of servers that **inetd**(8C) invokes when it receives an Internet request over a socket. Each server entry is composed of a single line of the form:

*service-name socket-type protocol wait-status uid server-program server-arguments*

Fields can be separated by either spaces or TAB characters. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search this file.

*service-name* is the name of a valid service listed in the file **/etc/services**. For RPC services, the value of the *service-name* field consists of the RPC service name, followed by a slash and either a version number or a range of version numbers (for example, **mountd/1**).

*socket-type* can be one of:

| | |
|---|---|
| **stream** | for a stream socket, |
| **dgram** | for a datagram socket, |
| **raw** | for a raw socket, |
| **rdm** | for a "reliably delivered message" socket, or |
| **seqpacket** | for a sequenced packet socket. |

*protocol* must be a recognized protocol listed in the file **/etc/protocols**. For RPC services, the field consists of the string "rpc" followed by a slash and the name of the protocol (for example, **rpc/udp** for an RPC service using the UDP protocol as a transport mechanism).

*wait-status* is **nowait** for all but "single-threaded" datagram servers — servers which do not release the socket until a timeout occurs (such as **comsat**(8C) and **talkd**(8C)). These must have the status **wait**. Although **tftpd**(8C) establishes separate "pseudo-connections", its forking behavior can lead to a race condition unless it is also given the status **wait**.

*uid* is the user ID under which the server should run. This allows servers to run with access privileges other than those for root.

*server-program* is either the pathname of a server program to be invoked by **inetd** to perform the requested service, or the value **internal** if **inetd** itself provides the service.

*server-arguments* If a server must be invoked with command-line arguments, the entire command line (including argument 0) must appear in this field (which consists of all remaining words in the entry). If the server expects **inetd** to pass it the address of its peer (for compatibility with 4.2BSD executable daemons), then the first argument to the command should be specified as '**%A**'.

## FILES

**/etc/inetd.conf**
**/etc/services**
**/etc/protocols**

## SEE ALSO

**services**(5), **comsat**(8C), **inetd**(8C), **talkd**(8C), **tftpd**(8C)

**NAME**

internat – key mapping table for internationalization

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

This file format is used for the file specified by the -f flag of **oldsetkeys**(1).

The file has three columns. First column is keytable identifier, one of: BASE, CTRL, SHIFT, CAPS, UP, BASE_ISO, SHIFT_ISO or ALTG. The second column is a decimal keystation number. The third column is hexadecimal keytable entry value. The file must end with line of "END, 0, 0". As usual, comment lines start with #.

**EXAMPLES**

This is the file for mapping keys to Canadian standards:

```
# /usr/lib/.setkeys: Key remapping, used by "setkeys remap"
#
# First column is keytable identifier:
#               BASE, CTRL, SHIFT, CAPS, UP, BASE_ISO, SHIFT_ISO or ALTG
# Second column is decimal keystation number
# Third column is hexadecimal keytable entry value
# File must end with line of "END, 0, 0"
# Comment lines must start with #
#
#
# --- Keymaps for Canadian keyboard ---
# > Define Alt Graph key (SHIFTKEYS+ALTGRAPH=86)
BASE 119 86
CTRL 119 86
SHIFT 119 86
CAPS 119 86
UP 119 86
# > Define Caps key (SHIFTKEYS+CAPSLOCK=80)
BASE 13 80
CTRL 13 80
SHIFT 13 80
CAPS 13 80
# > Define Floating Accent keys
#               FA_UMLAUT = A9
#               FA_CFLEX = AA
#               FA_TILDE = AB
#               FA_CEDILLA = AC
#               FA_ACUTE = AD
#               FA_GRAVE = AE
BASE 64 AA
SHIFT 64 A9
CAPS 64 A9
BASE 65 AC
SHIFT 65 AB
CAPS 65 AB
BASE 87 AE
SHIFT 87 AD
CAPS 87 AD
# > Define ASCII values
BASE 88 5B
```

```
SHIFT 88 7B
CAPS 88 7B
BASE 15 5D
SHIFT 15 7D
CAPS 15 7D
SHIFT 31 22
SHIFT 32 2F
SHIFT 35 3F
SHIFT 107 27
CAPS 107 27
SHIFT 108 60
CAPS 108 60
BASE 124 3C
SHIFT 124 3E
CAPS 124 3E
# > Define ISO values
BASE_ISO 109 E9
SHIFT_ISO 109 C9
# > Define Alternate Graph ISO values
ALTG 88 AB
ALTG 15 BB
ALTG 30 B1
ALTG 31 B2
ALTG 32 B3
ALTG 33 A2
ALTG 34 A4
ALTG 35 5E
ALTG 36 40
ALTG 37 A3
ALTG 38 5C
ALTG 40 AC
ALTG 41 23
ALTG 63 B6
ALTG 64 BC
ALTG 65 BD
ALTG 42 BE
ALTG 106 B5
ALTG 105 BA
# > End of file
END 0 0
```

**SEE ALSO**

**oldsetkeys**(1)

The *Sun386i Developer's Guide* for keystation number diagrams.

NAME

ipalloc.netrange – range of addresses to allocate

SYNOPSIS

**/etc/ipalloc.netrange**

AVAILABILITY

Sun386i systems only.

DESCRIPTION

This file, if it exists on the YP master of the **hosts.byaddr** YP map, specifies the ranges of IP addresses that can be allocated by the **ipallocd**(8C) daemon. This allows multiple address assignment authorities, probably in multiple administrative domains, to coexist on the same IP network by preallocating ranges of addresses. If the file does not exist, the daemon assumes that all addresses not listed in the **hosts** map may be freely allocated.

This file can contain blank lines. Comments begin with a # character and extend to the end of the current line. Ranges of free addresses are specified on one line per network or subnetwork.

The first token on the line is the IP address, in four part "dot" notation as also used in the **hosts** file, of the network or subnetwork described. It is separated from the second token by white space. The second token is a comma-separated list of local host number ranges on that network. These ranges take two forms: a single number specifies just that local host number, and two numbers separated by a dash specify all local host numbers starting at the first number and ending at the second. (In the case of a subnet, host numbers not in that subnet are excluded.)

For example, the following file would specify that a subset of the addresses on the class C network 192.9.200.0 may be allocated, and only some of the addresses on two particular subnets of the class B network 128.255.0.0 may be allocated. In any case, only non-broadcast addresses not listed in the **hosts** map are subject to allocation:

**# We have three network cables administered using automatic # IP address allocation.**

| | | | |
|---|---|---|---|
| **192.9.200.0** | **50-100,200-254** | **128.255.210.0** | **3,5,7,9,100-110** |
| **128.255.211.0** | **1-254** | | |

SEE ALSO

**hosts**(5), **netmasks**(5), **ipallocd**(8C)

BUGS

There is a silent limit of twenty ranges per network.

## NAME

link – link editor interfaces

## SYNOPSIS

**#include <link.h>**

## DESCRIPTION

Dynamically linked executables created by **ld**(1) contain a number of data structures that are used by the dynamic link editor to finish link-editing the program during program execution. These data structures are described with a **link_dynamic** structure, as defined in the **<link.h>** file. **ld** always identifies the location of this structure in the executable file with the symbol **__DYNAMIC**. This symbol is **ld**-defined and if referenced in an executable that does not require dynamic linking will have the value zero.

The program stub linked with "main" programs by compiler drivers such as **cc**(1) (called **crt0**) tests the definition of **__DYNAMIC** to determine whether or not the dynamic link editor should be invoked. Programs supplying a substitute for **crt0** must either duplicate this functionality or else require that the programs with which they are linked be linked *statically*. Otherwise, such replacement **crt0**'s must open and map in the executable **/usr/lib/ld.so** using **mmap**(2). Care should be taken to ensure that the expected mapping relationship between the "text" and "data" segments of the executable is maintained in the same manner that the **execve**(2) system call does. The first location following the **a.out** header of this executable is the entry point to a function that begins the dynamic link-editing process. This function must be called and supplied with two arguments. The first argument is an integer representing the revision level of the argument list, and should have the value "1". The second should be a pointer to an argument list structure of the form:

```
struct {
        int     crt_ba;             /* base address of ld.so */
        int     crt_dzfd;           /* open fd to /dev/zero */
        int     crt_ldfd;           /* open fd to ld.so */
        struct  link_dynamic *crt_dp; /* pointer to program's __DYNAMIC */
        char    **crt_ep;           /* environment strings */
        caddr_t crt_bp;             /* debugger hook */
}
```

The members of the structure are:

**crt_ba** The address at which **/usr/lib/ld.so** has been mapped.

**crt_dzfd** An open file descriptor for **/dev/zero**. **ld.so** will close this file descriptor before returning.

**crt_ldfd** The file descriptor used to map **/usr/lib/ld.so**. **ld.so** will close this file descriptor before returning.

**crt_dp** A pointer to the label **__DYNAMIC** in the executable which is calling **ld.so**.

**crt_ep** A pointer to the environment strings provided to the program.

**crt_bp** A location in the executable which contains an instruction that will be executed after the call to **ld.so** returns. This location is used as a breakpoint in programs that are being executed under the control of a debugger such as **adb**(1).

## SEE ALSO

**ld**(1), **mmap**(2), **a.out**(5)

## BUGS

These interfaces are under development and are subject to rapid change.

## NAME

magic – file command's magic number file

## DESCRIPTION

The **file**(1) command identifies the type of a file using, among other tests, a test for whether the file begins with a certain *magic number*. The file **/etc/magic** specifies what magic numbers are to be tested for, what message to print if a particular magic number is found, and additional information to extract from the file.

Each line of the file specifies a test to be performed. A test compares the data starting at a particular offset in the file with a 1-byte, 2-byte, or 4-byte numeric value or a string. If the test succeeds, a message is printed. The line consists of the following fields:

*offset* A number specifying the offset, in bytes, into the file of the data which is to be tested.

*type* The type of the data to be tested. The possible values are:

**byte** A one-byte value.

**short** A two-byte value.

**long** A four-byte value.

**string** A string of bytes.

The types **byte**, **short**, and **long** may optionally be followed by a mask specifier of the form *&number*. If a mask specifier is given, the value is AND'ed with the *number* before any comparisons are done. The *number* is specified in C form; for instance, **13** is decimal, **013** is octal, and **0x13** is hexadecimal.

**test** The value to be compared with the value from the file. If the type is numeric, this value is specified in **C** form; if it is a string, it is specified as a **C** string with the usual escapes permitted (for instance, \n for NEWLINE).

*Numeric values* may be preceded by a character indicating the operation to be performed. It may be '=', to specify that the value from the file must equal the specified value, '<', to specify that the value from the file must be less than the specified value, '>', to specify that the value from the file must be greater than the specified value, '&', to specify that all the bits in the specified value must be set in the value from the file, '^', to specify that at least one of the bits in the specified value must not be set in the value from the file, or **x** to specify that any value will match. If the character is omitted, it is assumed to be '='.

For string values, the byte string from the file must match the specified byte string; the byte string from the file which is matched is the same length as the specified byte string.

*message* The message to be printed if the comparison succeeds. If the string contains a **printf**(3S) format specification, the value from the file (with any specified masking performed) is printed using the message as the format string.

Some file formats contain additional information which is to be printed along with the file type. A line which begins with the character '>' indicates additional tests and messages to be printed. If the test on the line preceding the first line with a '>' succeeds, the tests specified in all the subsequent lines beginning with '>' are performed, and the messages printed if the tests succeed. The next line which does not begin with a '>' terminates this.

## FILES

**/etc/magic**

## SEE ALSO

**file**(1), **printf**(3S)

**BUGS**

There should be more than one level of subtests, with the level indicated by the number of '>' at the beginning of the line.

## NAME

mtab – mounted file system table

## SYNOPSIS

**/etc/mtab**

**#include <mntent.h>**

## DESCRIPTION

**mtab** resides in the **/etc** directory, and contains a table of filesystems currently mounted by the **mount**(8) command. **umount** removes entries from this file.

The file contains a line of information for each mounted filesystem, structurally identical to the contents of **/etc/fstab**, described in **fstab**(5). There are a number of lines of the form:

*fsname dir type opts freq passno*

for example:

**/dev/xy0a / 4.2 rw,noquota 1 2**

The file is accessed by programs using **getmntent**(3), and by the system administrator using a text editor.

## FILES

**/etc/mtab**
**/etc/fstab**

## SEE ALSO

**getmntent**(3), **fstab**(5), **mount**(8)

## NAME

netgroup – list of network groups

## DESCRIPTION

**netgroup** defines network wide groups, used for permission checking when doing remote mounts, remote logins, and remote shells. For remote mounts, the information in **netgroup** is used to classify machines; for remote logins and remote shells, it is used to classify users. Each line of the **netgroup** file defines a group and has the format

*groupname member1 member2 ....*

where member*i* is either another group name, or a triple:

(*hostname*, *username*, *domainname*)

Any of these three fields can be empty, in which case it signifies a wild card. Thus

**universal (,,)**

defines a group to which everyone belongs.

A gateway machine should be listed under all possible hostnames by which it may be recognized:

**wan (gateway,,) (gateway-ebb,,)**

Field names that begin with something other than a letter, digit or underscore (such as '–') work in precisely the opposite fashion. For example, consider the following entries:

**justmachines (analytica,–,sun)**
**justpeople (–,babbage,sun)**

The machine **analytica** belongs to the group **justmachines** in the domain **sun**, but no users belong to it. Similarly, the user **babbage** belongs to the group **justpeople** in the domain **sun**, but no machines belong to it.

The *domainname* field refers to the domain *n* which the triple is valid, not the name containing the trusted host.

## FILES

**/etc/netgroup**

## SEE ALSO

**getnetgrent**(3N), **exports**(5), **makedbm**(8), **ypserv**(8)

## NAME

netmasks – network mask data base

## DESCRIPTION

The **netmasks** file contains network masks used to implement IP standard subnetting. For each network that is subnetted, a single line should exist in this file with the network number, any number of SPACE or TAB characters, and the network mask to use on that network. Network numbers and masks may be specified in the conventional IP '.' notation (like IP host addresses, but with zeroes for the host part). For example,

**128.32.0.0 255.255.255.0**

can be used to specify the the Class B network 128.32.0.0 should have eight bits of subnet field and eight bits of host field, in addition to the standard sixteen bits in the network field. When running Yellow Pages, this file on the master is used for the **netmasks.byaddr** map.

## FILES

**/etc/netmasks**

## SEE ALSO

**ifconfig**(8C)

Postel, Jon, and Mogul, Jeff, *Internet Standard Subnetting Procedure*, RFC 950, Network Information Center, SRI International, Menlo Park, Calif., August 1985.

## NAME

netrc – file for ftp(1C) remote login data

## DESCRIPTION

The **.netrc** file contains data for logging in to a remote host over the network for file transfers by **ftp**(1C). This file resides in the user's home directory on the machine initiating the file transfer. Its permissions should be set to disallow read access by group and others (see **chmod**(1V)).

The following tokens are recognized; they may be separated by SPACE, TAB, or NEWLINE characters:

**machine***name*

Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the **ftp** command line or as an **open** command argument. Once a match is made, the subsequent **.netrc** tokens are processed, stopping when the EOF is reached or another **machine** token is encountered.

**login***name*

Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

**password***string*

Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note: if this token is present in the **.netrc** file, **ftp** will abort the auto-login process if the **.netrc** is readable by anyone besides the user.

**account***string*

Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

**macdef***name*

Define a macro. This token functions as the **ftp macdef** command functions. A macro is defined with the specified name; its contents begin with the next **.netrc** line and continue until a NULL line (consecutive NEWLINE characters) is encountered. If a macro named **init** is defined, it is automatically executed as the last step in the auto-login process.

## EXAMPLE

The command:

**machine ray login demo password mypassword**

allows an autologin to the machine **ray** using the login name **demo** with password **mypassword**.

## FILES

~**/.netrc**

## SEE ALSO

**chmod**(1V), **ftp**(1C), **ftpd**(8C)

**NAME**

networks – network name data base

**DESCRIPTION**

The **networks** file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional '.' notation using the **inet_network ()** routine from the Internet address manipulation library, **inet**(3N). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

**FILES**

**/etc/networks**

**SEE ALSO**

**getnetent**(3N), **inet**(3N)

**BUGS**

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

## NAME

passwd – password file

## SYNOPSIS

**/etc/passwd**

## DESCRIPTION

The **passwd** file contains basic information about each user's account. This file contains a one-line entry for each authorized user, of the form:

*username*:*password*:*uid*:*gid*:*gcos-field*:*home-dir*:*login-shell*

where

| | |
|---|---|
| *username* | is the user's login name. This field contains no uppercase characters, and must not be more than eight characters in length. |
| *password* | is the user's encrypted password, or a string of the form: *##name* if the encrypted password is in the **/etc/security/passwd.adjunct** file (see **passwd.adjunct**(5)). If this field is empty, **login**(1) does not request a password before logging the user in. |
| *uid* | is the user's numerical ID for the system, which must be unique. *uid* is generally a value between 0 and 32767. |
| *gid* | is the numerical ID of the group that the user belongs to. *gid* is generally a value between 0 an 32767. |
| *gcos-field* | is the user's real name, along with information to pass along in a mail-message heading. It is called the gcos-field for historical reasons. A & in this field stands for the login name (in cases where the login name appears in a user's real name). |
| *home-dir* | is the pathname to the directory in which the user is initially positioned upon logging in. |
| *login-shell* | is the user's initial shell program. If this field is empty, the default shell is **/usr/bin/sh**. |

The **passwd** file can also have lines beginning with a '+' (plus sign) which means to incorporate entries from the Yellow Pages. There are three styles of + entries in this file: by itself, + means to insert the entire contents of the Yellow Pages password file at that point; *+name* means to insert the entry (if any) for *name* from the Yellow Pages at that point; *+@netgroup* means to insert the entries for all members of the network group **netgroup** at that point. If a *+name* entry has a non-NULL *password*, *gcos*, *home-dir*, or *login-shell* field, the value of that field overrides what is contained in the Yellow Pages. The *uid* and *gid* fields cannot be overridden.

The **passwd** file can also have lines beginning with a '–' (minus sign) which means to disallow entries from the Yellow Pages. There are two styles of '–' entries in this file: *–name* means to disallow any subsequent entries (if any) for *name* (in this file or in the Yellow Pages); *–@netgroup* means to disallow any subsequent entries for all members of the network group *netgroup*.

The password file is an ASCII file that resides in the **/etc** directory. Because the encrypted passwords on a secure system are kept in the **passwd.adjunct** file, **/etc/passwd** has general read permission on all systems, and can be used by routines that map numerical user IDs to names.

Appropriate precautions must be taken to lock the **/etc/passwd** file against simultaneous changes if it is to be edited with a text editor; **vipw**(8) does the necessary locking.

EXAMPLE

Here is a sample **passwd** file when **passwd.adjunct** does not exist:

```
root:q.mJzTnu8icF.:0:10:God:/:/bin/csh
fred:6k/7KCFRPNVXg:508:10:% Fredericks:/usr2/fred:/bin/csh
+john:
+@documentation:no-login:
+::::Guest
```

Here is a sample **passwd** file when **passwd.adjunct** does exist:

```
root:##root:0:10:God:/:/bin/csh
fred:##fred:508:10:& Fredericks:/usr2/fred:/bin/csh
+john:
+@documentation:no-login:
+::::Guest
```

In this example, there are specific entries for users **root** and **fred**, to assure that they can log in even when the system is running standalone. The user **john** will have his password entry in the Yellow Pages incorporated without change; anyone in the netgroup **documentation** will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a gcos-field of **Guest**.

FILES

**/etc/passwd**
**/etc/security/passwd.adjunct**

SEE ALSO

**login**(1), **mail**(1), **passwd**(1), **crypt**(3), **getpwent**(3), **group**(5), **passwd.adjunct**(5), **adduser**(8), **sendmail**(8), **vipw**(8)

BUGS

**mail**(1) and **sendmail**(8) use the gcos-field to compose the **From:** line for addressing mail messages, but these programs get confused by nested parentheses when composing replies. This problem can be avoided by using different types of brackets within the gcos-field; for example:

**(& Fredricks [Podunk U <EE/CIS>] {818}-555-5555)**

## NAME

passwd.adjunct – user security data file

## SYNOPSIS

**/etc/security/passwd.adjunct**

## DESCRIPTION

The **passwd.adjunct** file contains the following information for each user:

| | |
|---|---|
| *name* | This is the user's login name in the system and it must be unique. |
| *password* | The encrypted password. |
| *minimum label* | The lowest security level at which this user is allowed to login (not used at C2 level). |
| *maximum label* | The highest security level at which this user is allowed to login (not used at C2 level). |
| *default label* | The security level at which this user will run unless a label is specified at login. |
| *always audit flags* | Flags specifying events always to be audited for this user's processes; see **audit_control**(5). |
| *never audit flags* | Flags specifying events never to be audited for this user's processes; see **audit_control**(5). |

Field are separated by a colon, and each user from the next by a NEWLINE.

The **passwd.adjunct** file can also have line beginning with a '+' (plus sign), which means to incorporate entries from the Yellow Pages. There are three styles of '+' entries: all by itself, '+' means to insert the entire contents of the Yellow Pages **passwd.adjunct** file at that point; *+name* means to insert the entry (if any) for *name* from the Yellow Pages at that point; *+@name* means to insert the entries for all members of the network group *name* at that point. If a '+' entry has a non-NULL password, it will override what is contained in the Yellow Pages.

## EXAMPLE

Here is a sample **/etc/security/passwd.adjunct** file:

```
root:q.mJzTnu8icF.:::::::::
ignatz:7KsI8CFRPNVXg::b,ap,bp,gp,dp,ic,r,d,l::+dc,+da:-dr:
rex:7HU8UUGRPNVXg:b,ap:b,ap,bp:b,bp::+ad:
+fred:9x.FFUw6xcJBa:::::::::
+:
```

The user **root** is the super-user, who has no special label constraints nor audit interest. The user **ignatz** may have any label from the lowest to the level **b** and any of a large number of categories. **ignatz** will run at system low unless he specifies otherwise. He is being audited on the system default event classes as well as data creations and access changes, but never for failed data reads. The user **rex** can function only at the level **b** and only in the categories **ap** or **ap** and **bp**. By default, he will run at '**b,bp**'. He is audited with the system defaults, except that successful administrative operations are not audited. The user **fred** will have the labels and audit flags that are specified in the Yellow Pages **passwd.adjunct** file. Any other users specified in the Yellow Pages will be able to log in on this system.

The user security data file resides in the **/etc/security** directory. Because it contains encrypted passwords, it does not have general read permission.

## FILES

**/etc/security/passwd.adjunct**
**/etc/security**

## SEE ALSO

**login**(1), **passwd**(1), **crypt**(3), **getpwaent**(3), **getpwent**(3), **audit_control**(5), **passwd**(5), **adduser**(8)

**NAME**

phones – remote host phone number data base

**SYNOPSIS**

**/etc/phones**

**DESCRIPTION**

The file **/etc/phones** contains the system-wide private phone numbers for the **tip**(1C) program. **/etc/phones** is normally unreadable, and so may contain privileged information. The format of **/etc/phones** is a series of lines of the form:

*<system-name>*[ \t]*<phone-number>*.

The system name is one of those defined in the **remote**(5) file and the phone number is constructed from **[0123456789–=*%]**. The '=' and '*' characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The '=' is required by the DF02-AC and the '*' is required by the BIZCOMP 1030.

Comment lines are lines containing a '#' sign in the first column of the line.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name **tip**(1C) will attempt to dial each one in turn, until it establishes a connection.

**FILES**

**/etc/phones**

**SEE ALSO**

**tip**(1C), **remote**(5)

**NAME**

plot – graphics interface

**DESCRIPTION**

Files of this format are produced by routines described in **plot**(3X), and are interpreted for various devices by commands described in **plot**(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the *x* and *y* values; each value is a signed integer. The last designated point in an *l*, *m*, *n*, or *p* instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in **plot**(3X).

**m** Move: the next four bytes give a new current point.

**n** Cont: draw a line from the current point to the point given by the next four bytes. See **plot**(1G).

**p** Point: plot the point given by the next four bytes.

**l** Line: draw a line from the point given by the next four bytes to the point given by the following four bytes.

**t** Label: place the following ASCII string so that its first character falls on the current point. The string is terminated by a NEWLINE.

**a** Arc: the first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

**c** Circle: the first four bytes give the center of the circle, the next two the radius.

**e** Erase: start another frame of output.

**f** Linemod: take the following string, up to a NEWLINE, as the style for drawing further lines. The styles are "dotted," "solid," "longdashed," "shortdashed," and "dotdashed." Effective only in **plot 4014** and **plot** *ver*.

**s** Space: the next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of **plot**(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

| | |
|---|---|
| 4014 | space(0, 0, 3120, 3120); |
| ver | space(0, 0, 2048, 2048); |
| 300, 300s | space(0, 0, 4096, 4096); |
| 450 | space(0, 0, 4096, 4096); |

**SEE ALSO**

**graph**(1G), **plot**(1G), **plot**(3X)

**NAME**

policies – network administration policies

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

The **policies** file contains information relevant to domain-wide administration policies. Each line contains two tokens, separated by white space; the first token is the name of an administrative policy, and the second is the value of that policy.

**FILES**

**/etc/policies**
**/var/yp/***domainname***/policies.{dir,pag}**

**SEE ALSO**

**pnpd**(8C), **rarpd**(8C), **logintool**(8)

**NAME**

printcap – printer capability data base

**SYNOPSIS**

**/etc/printcap**

**DESCRIPTION**

**printcap** is a simplified version of the **termcap**(5) data base for describing printers. The spooling system accesses the **printcap** file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base describes one printer. This data base may not be substituted for, as is possible for **termcap**, because it may allow accounting to be bypassed.

The default printer is normally **lp**, though the environment variable PRINTER may be used to override this. Each spooling utility supports a –P*printer* option to explicitly name a destination printer.

Refer to *System and Network Administration* for a discussion of how to set up the database for a given printer. On Sun386i systems, refer to **snap**(1) for information on setting up printers with the system and network administration program.

Each entry in the **printcap** file describes a printer, and is a line consisting of a number of fields separated by '**:**' characters. The first entry for each printer gives the names which are known for the printer, separated by '|' characters. The first name is conventionally a number. The second name given is the most common abbreviation for the printer, and the last name given should be a long name fully identifying the printer. The second name should contain no blanks; the last name may well contain blanks for readability. Entries may continue onto multiple lines by giving a '\' as the last character of a line, and empty fields may be included for readability.

Capabilities in **printcap** are all introduced by two-character codes, and are of three types:

*Boolean* Capabilities that indicate that the printer has some particular feature. Boolean capabilities are simply written between the '**:**' characters, and are indicated by the word 'bool' in the **type** column of the capabilities table below.

*Numeric* Capabilities that supply information such as baud-rates, number of lines per page, and so on. Numeric capabilities are indicated by the word **num** in the **type** column of the capabilities table below. Numeric capabilities are given by the two-character capability code followed by the '**#**' character, followed by the numeric value. For example:

**:br#1200:**

is a numeric entry stating that this printer should run at 1200 baud.

*String* Capabilities that give a sequence which can be used to perform particular printer operations such as cursor motion. String valued capabilities are indicated by the word **str** in the **type** column of the capabilities table below. String valued capabilities are given by the two-character capability code followed by an '**=**' sign and then a string ending at the next following '**:**'. For example,

**:rp=spinwriter:**

is a sample entry stating that the remote printer is named **spinwriter**.

**Sun386i DESCRIPTION**

On Sun386i systems, **lpr**(1) and related printing commands use the Yellow Pages name service to obtain the **printcap** entry for a named printer if the entry does not exist in the local **/etc/printcap** file. For example, when a user issues the command

**lpr -Pnewprinter foo**

**lpr** searches **/etc/printcap** on the local system for an entry for **newprinter**. If no local entry for **newprinter** exists, then **lpr** searches the YP map called **printcap**. The search is invisible to the user.

**lpr** creates the spooling directory for the printer automatically if no spooling directory exists.

System administrators can make a printer available to the entire YP domain by placing an entry for that printer in the YP **printcap** map, typically using **snap**. Otherwise, the system administrator must edit the **/etc/printcap** file on the YP master and then rebuild the YP map.

**CAPABILITIES**

| *Name* | *Type* | *Default* | *Description* |
|---|---|---|---|
| **af** | str | NULL | name of accounting file |
| **br** | num | none | if lp is a tty, set the baud rate (ioctl call) |
| **cf** | str | NULL | cifplot data filter |
| **df** | str | NULL | TeX data filter (DVI format) |
| **du** | str | 0 | User ID of user 'daemon'. |
| **fc** | num | 0 | if lp is a tty, clear flag bits |
| **ff** | str | "\f" | string to send for a form feed |
| **fo** | bool | false | print a form feed when device is opened |
| **fs** | num | 0 | like 'fc' but set bits |
| **gf** | str | NULL | graph data filter (plot(3X) format) |
| **hl** | bool | false | print the burst header page last |
| **ic** | bool | false | driver supports (non standard) ioctl to indent printout |
| **if** | str | NULL | name of text filter which does accounting |
| **lf** | str | "/dev/console" | error logging file name |
| **lo** | str | "lock" | name of lock file |
| **lp** | str | "/dev/lp" | device name to open for output |
| **mc** | num | 0 | maximum number of copies |
| **ms** | str | NULL | list of terminal modes to set or clear |
| **mx** | num | 1000 | maximum file size (in BUFSIZ blocks), zero = unlimited |
| **nd** | str | NULL | next directory for list of queues (unimplemented) |
| **nf** | str | NULL | ditroff data filter (device independent troff) |
| **of** | str | NULL | name of output filtering program |
| **pc** | num | 200 | price per foot or page in hundredths of cents |
| **pl** | num | 66 | page length (in lines) |
| **pw** | num | 132 | page width (in characters) |
| **px** | num | 0 | page width in pixels (horizontal) |
| **py** | num | 0 | page length in pixels (vertical) |
| **rf** | str | NULL | filter for printing FORTRAN style text files |
| **rg** | str | NULL | restricted group. Only members of group allowed access |
| **rm** | str | NULL | machine name for remote printer |
| **rp** | str | "lp" | remote printer name argument |
| **rs** | bool | false | restrict remote users to those with local accounts |
| **rw** | bool | false | open printer device read/write instead of read-only |
| **sb** | bool | false | short banner (one line only) |
| **sc** | bool | false | suppress multiple copies |
| **sd** | str | "/var/spool/lpd" | spool directory |
| **sf** | bool | false | suppress form feeds |
| **sh** | bool | false | suppress printing of burst page header |
| **st** | str | "status" | status file name |
| **tc** | str | NULL | name of similar printer; must be last |
| **tf** | str | NULL | troff data filter (C/A/T phototypesetter) |
| **tr** | str | NULL | trailer string to print when queue empties |
| **vf** | str | NULL | raster image filter |
| **xc** | num | 0 | if lp is a tty, clear local mode bits |
| **xs** | num | 0 | like 'xc' but set bits |

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

Note: the **fs**, **fc**, **xs**, and **xc** fields are flag *masks* rather than flag *values*. Certain default device flags are set when the device is opened by the line printer daemon if the device is connected to a terminal port. The flags indicated in the **fc** field are then cleared; the flags in the **fs** field are then set (or vice-versa, depending on the order of **fc***#nnnn* and **fs***#nnnn* in the **/etc/printcap** file). The bits cleared by the **fc** field and set by the **fs** field are those in the **sg_flags** field of the **sgtty** structure, as set by the **TIOCSETP ioctl** call, and the bits cleared by the **xc** field and set by the **xs** field are those in the "local flags" word, as set by the **TIOCLSET ioctl** call. See **ttcompat**(4M) for a description of these flags. For example, to set exactly the flags 06300 in the **fs** field, which specifies that the **EVENP**, **ODDP**, and **XTABS** modes are to be set, and all other flags are to be cleared, do:

**:fc#0177777:fs#06300:**

The same process applies to the **xc** and **xs** fields. Alternatively, the **ms** field can be used to specify modes to be set and cleared. These modes are specified as **stty**(1V) modes; any mode supported by **stty** may be specified, except for the baud rate which must be specified with the **br** field. This permits modes not supported by the older terminal interface described in **ttcompat**(4M) to be set or cleared. Thus, to set the terminal port to which the printer is attached to even parity, tab expansion, no newline to carriage-return/line-feed translation, and RTS/CTS flow control enabled, do:

**:ms=evenp,-tabs,nl,crtscts:**

On Sun386i systems, the **tc** field, as in the **termcap**(5) file, must appear last in the list of capabilities. It is recommended that each type of printer have a general entry describing common capabilities; then an individual printer can be defined with its particular capabilities plus a **tc** field that points to the general entry for that type of printer.

## FILES

**/etc/printcap**

## SEE ALSO

**lpq**(1), **lpr**(1), **lprm**(1), **snap**(1), **stty**(1V), **plot**(3X), **ttcompat**(4M), **termcap**(5), **lpc**(8), **lpd**(8), **pac**(8)

*System and Network Administration*

## NAME

protocols – protocol name data base

## SYNOPSIS

**/etc/protocols**

## DESCRIPTION

The **protocols** file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

*official protocol name*
*protocol number*
*aliases*

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

## EXAMPLE

The following example is taken from SunOS.

```
#
# Internet (IP) protocols
#
ip      0     IP      # internet protocol, pseudo protocol number
icmp    1     ICMP    # internet control message protocol
ggp     3     GGP     # gateway-gateway protocol
tcp     6     TCP     # transmission control protocol
pup     12    PUP     # PARC universal packet protocol
udp     17    UDP     # user datagram protocol
```

## FILES

**/etc/protocols**

## SEE ALSO

**getprotoent**(3N)

## BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

## NAME

publickey – public key database

## SYNOPSIS

**/etc/publickey**

## DESCRIPTION

**/etc/publickey** is the public key database used for secure networking. Each entry in the database consists of a network user name (which may either refer to a user or a hostname), followed by the user's public key (in hex notation), a colon, and then the user's secret key encrypted with its login password (also in hex notation).

This file is altered either by the user through the **chkey**(1) command or by the system administrator through the **newkey**(8) command. The file **/etc/publickey** should only contain data on the Yellow Pages master machine, where it is converted into the YP database **publickey.byname**.

## SEE ALSO

**chkey**(1), **publickey**(3R), **newkey**(8), **ypupdated**(8C)

**NAME**

queuedefs – queue description file for at, batch, and cron

**SYNOPSIS**

**/var/spool/cron/queuedefs**

**DESCRIPTION**

The **queuedefs** file describes the characteristics of the queues managed by **cron**(8). Each non-comment line in this file describes one queue. The format of the lines are as follows:

*q*.[*njob***j**][*nice***n**][*nwait***w**]

The fields in this line are:

*q* The name of the queue. **a** is the default queue for jobs started by **at**(1); **b** is the default queue for jobs started by **batch** (see **at**(1)); **c** is the default queue for jobs run from a **crontab**(5) file.

*njob* The maximum number of jobs that can be run simultaneously in that queue; if more than *njob* jobs are ready to run, only the first *njob* jobs will be run, and the others will be run as jobs that are currently running terminate. The default value is 100.

*nice* The **nice**(1) value to give to all jobs in that queue that are not run with a user ID of super-user. The default value is 2.

*nwait* The number of seconds to wait before rescheduling a job that was deferred because more than *njob* jobs were running in that job's queue, or because more than 25 jobs were running in all the queues. The default value is 60.

Lines beginning with # are comments, and are ignored.

**EXAMPLE**

```
#
# @(#)queuedefs 1.1 87/02/18 SMI; from S5R3
#
a.4j1n
b.2j2n90w
```

This file specifies that the **a** queue, for **at** jobs, can have up to 4 jobs running simultaneously; those jobs will be run with a **nice** value of 1. As no *nwait* value was given, if a job cannot be run because too many other jobs are running **cron** will wait 60 seconds before trying again to run it. The **b** queue, for **batch** jobs, can have up to 2 jobs running simultaneously; those jobs will be run with a **nice** value of 2. If a job cannot be run because too many other jobs are running, **cron** will wait 90 seconds before trying again to run it. All other queues can have up to 100 jobs running simultaneously; they will be run with a **nice** value of 2, and if a job cannot be run because too many other jobs are running **cron** will wait 60 seconds before trying again to run it.

**FILES**

**/var/spool/cron/queuedefs**

**SEE ALSO**

**at**(1), **nice**(1), **crontab**(5), **cron**(8)

**NAME**

rasterfile – Sun's file format for raster images

**SYNOPSIS**

**#include <rasterfile.h>**

**DESCRIPTION**

A rasterfile is composed of three parts: first, a header containing 8 integers; second, a (possibly empty) set of colormap values; and third, the pixel image, stored a line at a time, in increasing *y* order. The image is layed out in the file as in a memory pixrect. Each line of the image is rounded up to the nearest 16 bits.

The header is defined by the following structure:

```
struct rasterfile {
        int     ras_magic;
        int     ras_width;
        int     ras_height;
        int     ras_depth;
        int     ras_length;
        int     ras_type;
        int     ras_maptype;
        int     ras_maplength;
};
```

The *ras_magic* field always contains the following constant:

```
#define RAS_MAGIC     0x59a66a95
```

The *ras_width*, *ras_height*, and *ras_depth* fields contain the image's width and height in pixels, and its depth in bits per pixel, respectively. The depth is either 1 or 8, corresponding to standard frame buffer depths. The *ras_length* field contains the length in bytes of the image data. For an unencoded image, this number is computable from the *ras_width*, *ras_height*, and *ras_depth* fields, but for an encoded image it must be explicitly stored in order to be available without decoding the image itself. Note: the length of the header and of the (possibly empty) colormap values are not included in the value of the *ras_length* field; it is only the image data length. For historical reasons, files of type RT_OLD will usually have a 0 in the *ras_length* field, and software expecting to encounter such files should be prepared to compute the actual image data length if needed. The *ras_maptype* and *ras_maplength* fields contain the type and length in bytes of the colormap values, respectively. If *ras_maptype* is not RMT_NONE and the *ras_maplength* is not 0, then the colormap values are the *ras_maplength* bytes immediately after the header. These values are either uninterpreted bytes (usually with the *ras_maptype* set to RMT_RAW) or the equal length red, green and blue vectors, in that order (when the *ras_maptype* is RMT_EQUAL_RGB). In the latter case, the *ras_maplength* must be three times the size in bytes of any one of the vectors.

**FILES**

**/usr/include/rasterfile.h**

**SEE ALSO**

*SunView 1 Programmer's Guide*

## NAME

remote – remote host description file

## SYNOPSIS

**/etc/remote**

## DESCRIPTION

The systems known by **tip**(1C) and their attributes are stored in an ASCII file which is structured somewhat like the **termcap**(5) file. Each line in the file provides a description for a single *system*. Fields are separated by a colon ':'. Lines ending in a '\' character with an immediately following NEWLINE are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an '=' sign indicates a string value follows. A field name followed by a '#' sign indicates a following numeric value.

Entries named **tip*** and **cu*** are used as default entries by **tip**, and the **cu** interface to **tip**, as follows. When **tip** is invoked with only a phone number, it looks for an entry of the form **tip300**, where 300 is the baud rate with which the connection is to be made. When the **cu** interface is used, entries of the form **cu300** are used.

## CAPABILITIES

Capabilities are either strings (str), numbers (num), or boolean flags (bool). A string capability is specified by *capability=value*; for example, '**dv=/dev/harris**'. A numeric capability is specified by *capability#value*; for example, '**xa#99**'. A boolean capability is specified by simply listing the capability.

**at** **(str)** Auto call unit type.

**br** **(num)** The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is 300 baud.

**cm** **(str)** An initial connection message to be sent to the remote host. For example, if a host is reached through port selector, this might be set to the appropriate sequence required to switch to the host.

**cu** **(str)** Call unit if making a phone call. Default is the same as the *dv* field.

**di** **(str)** Disconnect message sent to the host when a disconnect is requested by the user.

**du** **(bool)** This host is on a dial-up line.

**dv** **(str)** **device(s)toopentoestablish** If this file refers to a terminal line, **tip**(1C) attempts to perform an exclusive open on the device to insure only one user at a time has access to the port.

**el** **(str)** Characters marking an end-of-line. The default is NULL. **tip** only recognizes '~' escapes after one of the characters in **el**, or after a carriage-return.

**fs** **(str)** Frame size for transfers. The default frame size is equal to BUFSIZ.

**hd** **(bool)** The host uses half-duplex communication, local echo should be performed.

**ie** **(str)** Input EOF marks. The default is NULL.

**oe** **(str)** Output EOF string. The default is NULL. When **tip** is transferring a file, this string is sent at EOF.

**pa** **(str)** The type of parity to use when sending data to the host. This may be one of "even", "odd", "none", "zero" (always set bit 8 to zero), "one" (always set bit 8 to 1). The default is "none".

**pn** **(str)** Telephone number(s) for this host. If the telephone number field contains an '@' sign, **tip** searches the **/etc/phones** file for a list of telephone numbers — see **phones**(5). A '%' sign in the telephone number indicates a 5-second delay for the Ventel Modem.

**tc** **(str)** Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature:

```
UNIX-1200:\
        :dv=/dev/cua0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:
arpavax|ax:\
        :pn=7654321%:tc=UNIX-1200
```

**FILES**

**/etc/remote**
**/etc/phones**

**SEE ALSO**

**tip**(1C), **phones**(5), **termcap**(5)

## NAME

resolve.conf – configuration file for name server routines

## DESCRIPTION

The resolver configuration file contains information that is read by the resolver routines the first time they are invoked in a process. The file is designed to be human readable and contains a list of name-value pairs that provide various types of resolver information.

The different configuration options are:

**nameserver** *address* — The Internet address (in dot notation) of a name server that the resolver should query. At least one name server should be listed. Up to MAXNS (currently 3) name servers may be listed, in that case the resolver library queries tries them in the order listed. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made).

**domain** *name* — The default domain to append to names that do not have a dot in them. This defaults to the domain set by the **domainname**(1) command.

**address** *address* — An Internet address (in dot notation) of any preferred networks. The list of addresses returned by the resolver will be sorted to put any addresses on this network before any others.

The name value pair must appear on a single line, and the keyword (for instance, **nameserver**) must start the line. The value follows the keyword, separated by white space.

## FILES

**/etc/resolve.conf**

## SEE ALSO

**domainname**(1), **gethostent**(3N), **resolver**(3), **named**(8C)

**NAME**

rgb – available colors (by name) for coloredit

**SYNOPSIS**

**.rgb**

**$HOME/.rgb**

**/usr/lib/.rgb**

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

**.rgb** is an ASCII file containing consecutive lines terminated by newlines. Each line starts with three integers, each in the range 0-255. These integers are the RGB equivilent for the color named on the same line. At least one tab character delimits the last integer from the name field. The coloreditor searches for this file, first in the current directory; next, in the users home directory; and finally, in **/usr/lib**. The user can add to or delete from the **.rgb** file that he or she has access to, thus changing the available color table for subsequent invocations of coloredit.

**EXAMPLE**

| | |
|---|---|
| 0 0 0 | Black |
| 0 0 255 | Blue |
| 95 159 159 | Cadet Blue |
| 66 66 111 | Cornflower Blue |
| 107 35 142 | Dark Slate Blue |

**SEE ALSO**

**coloredit**(1)

NAME

rpc – rpc program number data base

SYNOPSIS

**/etc/rpc**

DESCRIPTION

The *rpc* file contains user readable names that can be used in place of rpc program numbers. Each line has the following information:

name of server for the rpc program
rpc program number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Here is an example of the **/etc/rpc** file from the SunOS System.

```
#
#             rpc  1.10  87/04/10
#
portmapper    100000        portmap sunrpc
rstatd                      100001        rstat rup perfmeter
rusersd                     100002        rusers
nfs                         100003        nfsprog
ypserv                      100004        ypprog
mountd                      100005        mount showmount
ypbind                      100007
walld                       100008        rwall shutdown
yppasswdd     100009        yppasswd
etherstatd    100010        etherstat
rquotad                     100011        rquotaprog quota rquota
sprayd                      100012        spray
3270_mapper   100013
rje_mapper    100014
selection_svc 100015        selnsvc
database_svc  100016
rexd                        100017        rex
alis                        100018
sched                       100019
llockmgr      100020
nlockmgr      100021
x25.inr                     100022
statmon                     100023
status                      100024
bootparam     100026
ypupdated     100028        ypupdate
keyserv                     100029        keyserver
```

FILES

**/etc/rpc**

SEE ALSO

**getrpcent**(3N)

**NAME**

sccsfile – format of SCCS file

**DESCRIPTION**

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be represented graphically as '**@**'. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form **DDDDD** represent a five digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

*Checksum*

The checksum is the first line of an SCCS file. The form of the line is:

**@hDDDDD**

The value of the checksum is the sum of all characters, except those of the first line. The **@h** provides a *magic number* of (octal) 064001.

*Delta table*

The delta table consists of a variable number of entries of the form:

**@s DDDDD/DDDDD/DDDDD**
**@d** <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> **DDDDD DDDDD**
**@i DDDDD ...**
**@x DDDDD ...**
**@g DDDDD ...**
**@m** <MR *number*>
.
.
.
**@c** <*comments*> ...
.
.
.
**@e**

The first line (@s) contains the number of lines inserted/deleted/unchanged respectively. The second line (**@d**) contains the type of the delta (currently, normal: **D**< and removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The **@i**, **@x**, and **@g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The **@m** lines (optional) each contain one MR number associated with the delta; the @c lines contain comments associated with the delta.

The **@e** line ends the delta table entry.

*User names*

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by NEWLINE characters. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **@u** and @U. An empty list allows anyone to

make a delta.

*Flags*

Keywords used internally (see **admin**(1) for more information on their use). Each flag line takes the form:

@f <*flag*> <*optional text*>

The following flags are defined:

| | |
|---|---|
| @f t | <*type of program*> |
| @f v | <*program name*> |
| @f i | |
| @f b | |
| @f m | <*module name*> |
| @f f | <*floor*> |
| @f c | <*ceiling*> |
| @f d | <*default-sid*> |
| @f n | |
| @f j | |
| @f l | <*lock-releases*> |
| @f q | <*user defined*> |
| @f e | <**0**\|**1**> |

The **t** flag defines the replacement for the **identification keyword.** The **v** flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The **i** flag controls the warning/error aspect of the '**No id keywords**' message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the **b** flag is present the **–b** keyletter may be used on the **get** command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the **sccsfile.5** identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a **get** command. The **n** flag causes **delta** to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes **get** to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing (**get**(1) with the **–e** keyletter). The **q** flag defines the replacement for the **identification keyword.** The **e** flag indicates whether a file is encoded or not. A **1** indicates that the file is encoded. Files need to be encoded when they contain control characters, or when they do not end with a NEWLINE. The **e** flag allows for any type of file to be checked in.

*Comments*

Arbitrary text surrounded by the bracketing lines **@t** and **@T**. The comments section typically will contain a description of the file's purpose.

*Body*

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, represented by:

**@I DDDDD**
**@D DDDDD**
**@E DDDDD**

respectively. The digit string is the serial number corresponding to the delta for the control line.

**SEE ALSO**

**admin**(1), **delta**(1), **get**(1), **prs(1)**

**NAME**

services – Internet services and aliases

**DESCRIPTION**

The **services** file contains an entry for each service available through the DARPA Internet. Each entry consists of a line of the form:

*service-name port/protocol aliases*

| | |
|---|---|
| *service-name* | This is the official Internet service name. |
| *port/protocol* | This field is composed of the port number and protocol through which the service is provided (for instance, **512/tcp**). |
| **aliases** | This is a list of alternate names by which the service might be requested. |

Fields can be separated by any number of spaces or TAB's. A '#' (pound-sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

**FILES**

**/etc/services**

**SEE ALSO**

**getservent**(3N), **inetd.conf**(5)

**BUGS**

A name server should be used instead of a static file.

## NAME

sm, sm.bak, sm.state – in.statd directory and file structures

## SYNOPSIS

**/etc/sm, /etc/sm.bak, /etc/sm.state**

## DESCRIPTION

**/etc/sm** and **/etc/sm.bak** are directories generated by **in.statd**. Each entry in **/etc/sm** represents the name of the machine to be monitored by the **in.statd** daemon. Each entry in **/etc/sm.bak** represents the name of the machine to be notified by the **in.statd** daemon upon its recovery.

**/etc/sm.state** is a file generated by **rpc.statd** to record the its version number. This version number is incremented each time a crash or recovery takes place.

## FILES

**/etc/sm**
**/etc/sm.bak**
**/etc/sm.state**

## SEE ALSO

**lockd**(8C), **statd**(8C)

**NAME**

sm, sm.bak, state – statd directories and file structures

**SYNOPSIS**

**/etc/sm /etc/sm.bak /etc/state**

**DESCRIPTION**

**/etc/sm** and **/etc/sm.bak** are directories generated by **statd**. Each entry in **/etc/sm** represents the name of the machine to be monitored by the **statd** daemon. Each entry in **/etc/sm.bak** represents the name of the machine to be notified by the **statd** daemon upon its recovery.

**/etc/state** is a file generated by **statd** to record its version number. This version number is incremented each time a crash or recovery takes place.

**FILES**

**/etc/sm**
**/etc/sm.bak**
**/etc/state**

**SEE ALSO**

**lockd**(8C), **statd**(8C)

## NAME

syslog.conf – configuration file for syslogd system log daemon

## SYNOPSIS

**/etc/syslog.conf**

## DESCRIPTION

The file **/etc/syslog.conf** contains information used by the system log daemon, **syslogd**(8), to forward a system message to appropriate log files and/or users. **syslog** preprocesses this file through **m4**(1V) to obtain the correct information for certain log files.

A configuration entry is composed of two TAB-separated fields:

*selector* *action*

The *selector* field contains a semicolon-separated list of priority specifications of the form:

*facility.level*[;*facility.level*]

where *facility* is a system facility, or comma-separated list of facilities, and *level* is an indication of the severity of the condition being logged. Recognized values for *facility* include:

**user** Messages generated by user processes. This is the default priority for messages from programs or facilities not listed in this file.

**kern** Messages generated by the kernel.

**mail** The mail system.

**daemon** System daemons, such as **ftpd**(8C), **routed**(8C), etc.

**auth** The authorization system: **login**(1), **su**(1), **getty**(8), etc.

**lpr** The line printer spooling system: **lpr**(1), **lpc**(8), **lpd**(8), etc.

**news** Reserved for the USENET network news system.

**uucp** Reserved for the UUCP system; it does not currently use the **syslog** mechanism.

**cron** The **cron/at** facility; **crontab**(1), **at**(1), **cron**(8), etc.

**local0-7** Reserved for local use.

**mark** For timestamp messages produced internally by **syslogd**.

***** An asterisk indicates all facilities except for the **mark** facility.

Recognized values for *level* are (in descending order of severity):

**emerg** For panic conditions that would normally be broadcast to all users.

**alert** For conditions that should be corrected immediately, such as a corrupted system database.

**crit** For warnings about critical conditions, such as hard device errors.

**err** For other errors.

**warning** For warning messages.

**notice** For conditions that are not error conditions, but may require special handling.

**info** Informational messages.

**debug** For messages that are normally used only when debugging a program.

**none** Do not send messages from the indicated *facility* to the selected file. For example, a *selector* of

***.debug;mail.none**

will send all messages *except* mail messages to the selected file.

The *action* field indicates where to forward the message. Values for this field can have one of four forms:

- A filename, beginning with a leading slash, which indicates that messages specified by the *selector* are to be written to the specified file. The file will be opened in append mode.
- The name of a remote host, prefixed with an **@**, as with: *@server*, which indicates that messages specified by the *selector* are to be forwarded to the **syslogd** on the named host.
- A comma-separated list of usernames, which indicates that messages specified by the *selector* are to be written to the named users if they are logged in.
- An asterisk, which indicates that messages messages specified by the *selector* are to be written to all logged-in users.

Blank lines are ignored. Lines for which the first nonwhite character is a '#' are treated as comments.

**Sun386i DESCRIPTION**

The file is as described above, except that there is an additional valid entry type, for translation. A line containing the keyword "translate," if present, specifies how system error messages are translated, suppressed, or forwarded to appropriate log files and/or users.

A translation entry in the file is composed of five TAB-separated fields:

*translate* *source* *facility* *input* *output*

The *translate* field consists of the word **translate** and is used to indicate that this is a translation entry.

The *source* field contains a comma separated list of source names. Recognized sources are:

| | |
|---|---|
| **klog** | Messages placed in **/dev/klog** by the kernel. |
| **log** | Messages placed in **/dev/log** file by local programs. |
| **syslog** | Messages placed in the internet socket by programs on other systems. |
| * | An asterisk indicates all three sources (**klog**, **log** and **syslog**). |

The *facility* field contains a comma-separated list of facilities.

The *input* field is the name of the file used to map error messages (in printf format strings) to numbers. This number is used to locate a new string in the file specified in the output field. The format of both files is described in **translate**(5).

The output file specified by the output field translates the numbers from the input file into the desired error messages, and also specifies the format to be used to output each message.

**EXAMPLE**

With the following configuration file:

```
*.notice;mail.info      /var/log/notice
*.crit                  /var/log/critical
kern,mark.debug         /dev/console
kern.err                @server
*.emerg                 *
*.alert                 root,operator
*.alert;auth.warning    /var/log/auth
```

**syslogd** will log all mail system messages except **debug** messages and all **notice** (or higher) messages into a file named **/var/log/notice**. It logs all critical messages into **/var/log/critical**, and all kernel messages and 20-minute marks onto the system console.

Kernel messages of **err** (error) severity or higher are forwarded to the machine named *server*. Emergency messages are forwarded to all users. The users "root" and "operator" are informed of any **alert** messages. All messages from the authorization system of **warning** level or higher are logged in the file **/var/log/auth**.

FILES

**/etc/syslog.conf**
**/var/log/notice**
**/var/log/critical**
**/var/log/auth**

SEE ALSO

**at**(1), **crontab**(1), **logger**(1), **login**(1), **lpr**(1), **m4**(1V), **su**(1), **syslog**(3), **translate**(5), **cron**(8), **ftpd**(8C), **getty**(8), **lpc**(8), **lpd**(8), **routed**(8C), **syslogd**(8)

**NAME**

tar – tape archive file format

**DESCRIPTION**

**tar**, (the tape archive command) dumps several files into one, in a medium suitable for transportation.

A "tar tape" or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an EOF indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the **tar**(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads, unless the **B** keyletter is used.

The header block looks like:

```
#define TBLOCK 512
#define NAMSIZ 100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
        } dbuf;
};
```

*name* is a NULL-terminated string. The other fields are zero-filled octal numbers in ASCII. Each field (of width *w*) contains w-2 digits, a SPACE, and a NULL, except *size* and *mtime*, which do not contain the trailing NULL. *name* is the name of the file, as specified on the **tar** command line. Files dumped because they were in a directory which was named in the command line have the directory name as prefix and */filename* as suffix. *mode* is the file mode, with the top bit masked off. *uid* and *gid* are the user and group numbers which own the file. *size* is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. *mtime* is the modification time of the file at the time it was dumped. *chksum* is a decimal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *linkflag* is ASCII '0' if the file is "normal" or a special file, ASCII '1' if it is an hard link, and ASCII '2' if it is a symbolic link. The name linked-to, if any, is in *linkname*, with a trailing NULL. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given inode number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

The encoding of the header is designed to be portable across machines.

**SEE ALSO**

**tar**(1)

**BUGS**

Names or linknames longer than NAMSIZ produce error reports and cannot be dumped.

NAME

term – terminal driving tables for nroff

SYNOPSIS

**/usr/lib/term/tab***name*

DESCRIPTION

**nroff**(1) uses driving tables to customize its output for various types of output devices, such as terminals, line printers, daisy-wheel printers, or special output filter programs. These driving tables are written as C programs, compiled, and installed in the directory **/usr/lib/term**. The *name* of the output device is specified with the **–T** option of **nroff**. The structure of the terminal table is as follows:

```
#define    INCH    240

struct {
        int bset;
        int breset;
        int Hor;
        int Vert;
        int Newline;
        int Char;
        int Em;
        int Halfline;
        int Adj;
        char *twinit;
        char *twrest;
        char *twnl;
        char *hlr;
        char *hlf;
        char *flr;
        char *bdon;
        char *bdoff;
        char *ploton;
        char *plotoff;
        char *up;
        char *down;
        char *right;
        char *left;
        char *codetab[256–32];
        char *zzz;
} t;
```

The meanings of the various fields are as follows:

**bset** Bits to set in the **sg_flags** field of the **sgtty** structure before output; see **ttcompat**(4M).

**breset** Bits to reset in the **sg_flags** field of the **sgtty** structure after output; see **ttcompat**(4M).

**Hor** Horizontal resolution in fractions of an inch.

**Vert** Vertical resolution in fractions of an inch.

**Newline** Space moved by a NEWLINE (LINEFEED) character in fractions of an inch.

**Char** Quantum of character sizes, in fractions of an inch. (that is, a character is a multiple of **Char** units wide)

**Em** Size of an em in fractions of an inch.

**Halfline** Space moved by a half-LINEFEED (or half-reverse-LINEFEED) character in fractions of an inch.

**Adj** Quantum of white space, in fractions of an inch. (that is, white spaces are a multiple of **Adj** units wide)

Note: if this is less than the size of the SPACE character (in units of **Char**; see below for how the sizes of characters are defined), **nroff** will output fractional SPACE characters using plot mode. Also, if the **–e** switch to **nroff** is used, **Adj** is set equal to **Hor** by **nroff**.

**twinit** Set of characters used to initialize the terminal in a mode suitable for **nroff**.

**twrest** Set of characters used to restore the terminal to normal mode.

**twnl** Set of characters used to move down one line.

**hlr** Set of characters used to move up one-half line.

**hlf** Set of characters used to move down one-half line.

**flr** Set of characters used to move up one line.

**bdon** Set of characters used to turn on hardware boldface mode, if any.

**bdoff** Set of characters used to turn off hardware boldface mode, if any.

**ploton** Set of characters used to turn on hardware plot mode (for Diablo type mechanisms), if any.

**plotoff** Set of characters used to turn off hardware plot mode (for Diablo type mechanisms), if any.

**up** Set of characters used to move up one resolution unit (**Vert**) in plot mode, if any.

**down** Set of characters used to move down one resolution unit (**Vert**) in plot mode, if any.

**right** Set of characters used to move right one resolution unit (**Hor**) in plot mode, if any.

**left** Set of characters used to move left one resolution unit (**Hor**) in plot mode, if any.

**codetab** Definition of characters needed to print an **nroff** character on the terminal. The first byte is the number of character units (**Char**) needed to hold the character; that is, **\001** is one unit wide, **\002** is two units wide, etc. The high-order bit (0200) is on if the character is to be underlined in underline mode (**.ul**). The rest of the bytes are the characters used to produce the character in question. If the character has the sign (0200) bit on, it is a code to move the terminal in plot mode. It is encoded as:

| | |
|---|---|
| 0100 bit on | vertical motion. |
| 0100 bit off | horizontal motion. |
| 040 bit on | negative (up or left) motion. |
| 040 bit off | positive (down or right) motion. |
| 037 bits | number of such motions to make. |

**zzz** A zero terminator at the end.

All quantities which are in units of fractions of an inch should be expressed as '**INCH***num*/*denom*', where *num* and *denom* are respectively the numerator and denominator of the fraction; that is, 1/48 of an inch would be written as '**INCH/48**'.

If any sequence of characters does not pertain to the output device, that sequence should be given as a null string.

The following is a sample **codetab** encoding.

```
"\001 ",                    /*space*/
"\001!",                    /*!*/
"\001\"",                   /*"*/
"\001#",                    /*#*/
"\001$",                    /*$*/
```

```
"\001%",                /*%*/
"\001&",                /*&*/
"\001'",                /*'*/
"\001(",                /*(*/
"\001)",                /*)*/
"\001*",                /***/
"\001+",                /*+*/
"\001,",                /*,*/
"\001-",                /*-*/
"\001.",                /*.*/
"\001/",                /*/*/
"\2010",                /*0*/
"\2011",                /*1*/
"\2012",                /*2*/
"\2013",                /*3*/
"\2014",                /*4*/
"\2015",                /*5*/
"\2016",                /*6*/
"\2017",                /*7*/
"\2018",                /*8*/
"\2019",                /*9*/
"\001:",                /*:*/
"\001;",                /*;*/
"\001<",                /*<*/
"\001=",                /*=*/
"\001>",                /*>*/
"\001?",                /*?*/
"\001@",                /*@*/
"\201A",                /*A*/
"\201B",                /*B*/
"\201C",                /*C*/
"\201D",                /*D*/
"\201E",                /*E*/
"\201F",                /*F*/
"\201G",                /*G*/
"\201H",                /*H*/
"\201I",                /*I*/
"\201J",                /*J*/
"\201K",                /*K*/
"\201L",                /*L*/
"\201M",                /*M*/
"\201N",                /*N*/
"\201O",                /*O*/
"\201P",                /*P*/
"\201Q",                /*Q*/
"\201R",                /*R*/
"\201S",                /*S*/
"\201T",                /*T*/
"\201U",                /*U*/
"\201V",                /*V*/
"\201W",                /*W*/
"\201X",                /*X*/
"\201Y",                /*Y*/
```

```
"\201Z",                    /*Z*/
"\001[",                    /*[*/
"\001\\",                   /*\*/
"\001]",                    /*]*/
"\001^",                    /*^*/
"\001_",                    /*_*/
"\001`",                    /*`*/
"\201a",                    /*a*/
"\201b",                    /*b*/
"\201c",                    /*c*/
"\201d",                    /*d*/
"\201e",                    /*e*/
"\201f",                    /*f*/
"\201g",                    /*g*/
"\201h",                    /*h*/
"\201i",                    /*i*/
"\201j",                    /*j*/
"\201k",                    /*k*/
"\201l",                    /*l*/
"\201m",                    /*m*/
"\201n",                    /*n*/
"\201o",                    /*o*/
"\201p",                    /*p*/
"\201q",                    /*q*/
"\201r",                    /*r*/
"\201s",                    /*s*/
"\201t",                    /*t*/
"\201u",                    /*u*/
"\201v",                    /*v*/
"\201w",                    /*w*/
"\201x",                    /*x*/
"\201y",                    /*y*/
"\201z",                    /*z*/
"\001{",                    /*{*/
"\001|",                    /*|*/
"\001}",                    /*}*/
"\001~",                    /*~*/
"\000\0",                   /*narrow sp*/
"\001-",                    /*hyphen*/
"\001\016Z\017",            /*bullet*/
"\002[]",                   /*square*/
"\002--",                   /*3/4 em dash*/
"\001_",                    /*rule*/
"\0031/4",                  /*1/4*/
"\0031/2",                  /*1/2*/
"\0033/4",                  /*3/4*/
"\001-",                    /*minus*/
"\202fi",                   /*fi*/
"\202fl",                   /*fl*/
"\202ff",                   /*ff*/
"\203ffi",                  /*ffi*/
"\203ffl",                  /*ffl*/
"\001\016p\017",            /*degree*/
```

```
"\001|\b\342-\302",     /*dagger*/
"\001\301s\343s\302",   /*section*/
"\001'",                /*foot mark*/
"\001\033Z",            /*acute accent*/
"\001`",                /*grave accent*/
"\001_",                /*underrule*/
"\001/",                /*long slash*/
"\000\0",               /*half narrow space*/
"\001 ",                /*unpaddable space*/
"\001\016A\017",        /*alpha*/
"\001\016B\017",        /*beta*/
"\001\016C\017",        /*gamma*/
"\001\016D\017",        /*delta*/
"\001\016E\017",        /*epsilon*/
"\001\016F\017",        /*zeta*/
"\001\016G\017",        /*eta*/
"\001\016H\017",        /*theta*/
"\001\016I\017",        /*iota*/
"\001\016J\017",        /*kappa*/
"\001\016K\017",        /*lambda*/
"\001\016L\017",        /*mu*/
"\001\016M\017",        /*nu*/
"\001\016N\017",        /*xi*/
"\001\016O\017",        /*omicron*/
"\001\016P\017",        /*pi*/
"\001\016Q\017",        /*rho*/
"\001\016R\017",        /*sigma*/
"\001\016S\017",        /*tau*/
"\001\016T\017",        /*upsilon*/
"\001\016U\017",        /*phi*/
"\001\016V\017",        /*chi*/
"\001\016W\017",        /*psi*/
"\001\016X\017",        /*omega*/
"\001\016#\017",        /*Gamma*/
"\001\016$\017",        /*Delta*/
"\001\016(\017",        /*Theta*/
"\001\016+\017",        /*Lambda*/
"\001\016.\017",        /*Xi*/
"\001\0160\017",        /*Pi*/
"\001\0169\017",        /*Sigma*/
"\000",                 /**/
"\001\0164\017",        /*Upsilon*/
"\001\0165\017",        /*Phi*/
"\001\0167\017",        /*Psi*/
"\001\0168\017",        /*Omega*/
"\001\016[\017",        /*square root*/
"\001\016Y\017",        /*\(ts yields script-l*/
"\001\016k\017",        /*root en*/
"\001>\b_",             /*>=*/
"\001<\b_",             /*<=*/
"\001=\b_",             /*identically equal*/
"\001-",                /*equation minus*/
"\001\016o\017",        /*approx =*/
```

```
"\001\016n\017",                                    /*approximates*/
"\001=\b/",                                         /*not equal*/
"\002-\242-\202>",                                  /*right arrow*/
"\002<\b\202-\242\200-",                            /*left arrow*/
"\001|\b^",                                         /*up arrow*/
"\001|\b\302v\342",                                 /*down arrow*/
"\001=",                                            /*equation equal*/
"\001\016|\017",                                    /*multiply*/
"\001\016}\017",                                    /*divide*/
"\001\016j\017",                                    /*plus-minus*/
"\001\243|\203_\203|\243",                          /*cup (union)*/
"\001\243|\203\351_\311\203|\243",                  /*cap (intersection)*/
"\001\243(\203\302-\345-\303",                      /*subset of*/
"\001\302-\345-\303\203)\243",                      /*superset of*/
"\001_\b\243(\203\302-\345-\303",                   /*improper subset*/
"\001_\b\302-\345-\303\203)\243",                   /*improper superset*/
"\001\016~\017",                                    /*infinity*/
"\001\200o\201\301‘\241\341‘\241\341‘\201\301",     /*partial derivative*/
"\001\016:\017",                                    /*gradient*/
"\001\200-\202\341,\301\242",                       /*not*/
"\001\016?\017",                                    /*integral sign*/
"\002o\242c\202",                                   /*proportional to*/
"\001O\b/",                                         /*empty set*/
"\001<\b\341-\302",                                 /*member of*/
"\001+",                                            /*equation plus*/
"\003(R)",                                          /*registered*/
"\003(C)",                                          /*copyright*/
"\001|",                                            /*box rule */
"\001\033Y",                                        /*cent sign*/
"\001|\b\342=\302",                                 /*double dagger*/
"\002=>",                                           /*right hand*/
"\002<=",                                           /*left hand*/
"\001*",                                            /*math * */
"\001\0162\017",                                    /*\(bs yields small sigma*/
"\001|",                                            /*or (was star)*/
"\001O",                                            /*circle*/
"\001|",                                            /*left top of big brace*/
"\001|",                                            /*left bot of big brace*/
"\001|",                                            /*right top of big brace*/
"\001|",                                            /*right bot of big brace*/
"\001\016]\017",                                    /*left center of big brace*/
"\001\016\\\017",                                   /*right center of big brace*/
"\001|",                                            /*bold vertical*/
"\001|",                                            /*left floor (lb of big bracket)*/
"\001|",                                            /*right floor (rb of big bracket)*/
"\001|",                                            /*left ceiling (lt of big bracket)*/
"\001|"                                             /*right ceiling (rt of big bracket)*/
```

FILES

| | |
|---|---|
| **/usr/lib/term/tab***name* | driving tables |
| **/usr/lib/term/README** | list of terminals supported by **nroff**(1) |

SEE ALSO

**nroff**(1), **ttcompat**(4M)

## NAME

term – format of compiled term file

## SYNOPSIS

**term**

## DESCRIPTION

Compiled **terminfo** descriptions are placed under the directory **/usr/share/lib/terminfo**. In order to avoid a linear search of a huge system directory, a two-level scheme is used: **/usr/share/lib/terminfo/c/name** where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, *act4* can be found in the file **/usr/share/lib/terminfo/a/act4**. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the **tic**(8V) program, and read by the routine **setupterm** (see **curses**(3V)). Both of these pieces of software are part of **curses**(3V). The file is divided into six parts:

the header,
terminal names,
boolean flags,
numbers,
strings,
and
string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are:

(1) the magic number (octal 0432);
(2) the size, in bytes, of the names section;
(3) the number of bytes in the boolean section;
(4) the number of short integers in the numbers section;
(5) the number of offsets (short integers) in the strings section;
(6) the size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value −1 is represented by 0377, 0377, other negative value are illegal. The −1 generally means that a capability is missing from this terminal. Note: this format corresponds to the hardware of the VAX and PDP-11. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the '|' character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. The capabilities are in the same order as the file **<term.h>**.

Between the boolean section and the number section, a NULL byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is −1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of −1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information **$<*nn*>** and parameter information ***%x*** are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is NULL terminated.

Note: it is possible for **setupterm** to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm** has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm** must be prepared for both possibilities — this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
cr=^M, cud1=^J, ind=^J, bel=^G, am, cub1=^H,
ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
cols#80, lines#24, cuf1=^X, cuu1=^Z, home=^],
```

```
000 032 001      \0 025  \0  \b  \0 212  \0   "  \0   m   i   c   r
020   o   t   e   r   m   |   a   c   t   4   |   m   i   c   r   o
040   t   e   r   m       a   c   t       i   v  \0  \0 001  \0  \0
060  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
100  \0  \0   P  \0 377 377 030  \0 377 377 377 377 377 377 377 377
120 377 377 377 377  \0  \0 002  \0 377 377 377 377 004  \0 006  \0
140  \b  \0 377 377 377 377  \n  \0 026  \0 030  \0 377 377 032  \0
160 377 377 377 377 034  \0 377 377 036  \0 377 377 377 377 377 377
200 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520 377 377 377 377      \0 377 377 377 377 377 377 377 377 377 377
540 377 377 377 377 377 377 007  \0  \r  \0  \f  \0 036  \0 037  \0
560 024   %   p   1   %   c   %   p   2   %   c  \0  \n  \0 035  \0
600  \b  \0 030  \0 032  \0  \n  \0
```

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

**FILES**

**/usr/share/lib/terminfo/*/***
compiled terminal capability data base

**SEE ALSO**

**curses**(3V), **terminfo**(5V), **tic**(8V)

## NAME

termcap – terminal capability data base

## DESCRIPTION

**termcap** is a data base describing the capabilities of terminals. Terminals are described in **termcap** source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs such as **vi**(1), and libraries such as **curses**(3X), so they can work with a variety of terminals without changes to the programs.

Each **termcap** entry consist of a number of colon-separated (**:**) fields. The first field for each terminal lists the various names by which it is known, separated by bar ( | ) characters. The first name is always two characters long, and is used by older (version 6) systems (which store the terminal type in a 16-bit word in a system-wide database). The second name given is the most common abbreviation for the terminal (this is the one to which the environment variable **TERM** would normally be set). The last name should fully identify the terminal's make and model. All other names are taken as synonyms for the initial terminal name. All names but the first and last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for added readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions:

- The particular piece of hardware making up the terminal should have a root name chosen; for example, for the Hewlett-Packard 2621, **hp2621**. This name should not contain hyphens.
- Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, a **vt100** in 132-column mode would be given as: **vt100–w**. The following suffixes should be used where possible:

| *Suffix* | *Meaning* | *Example* |
|---|---|---|
| **–w** | wide mode (more than 80 columns) | **vt100–w** |
| **–am** | with automatic margins (usually default) | **vt100–am** |
| **–nam** | without automatic margins | **vt100–nam** |
| *–n* | number of lines on the screen | **aaa–60** |
| **–na** | no arrow keys (leave them in local) | **concept100–na** |
| *–n***p** | number of pages of memory | **concept100–4p** |
| **–rv** | reverse video | **concept100–rv** |

Terminal entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with **#**.

### Types of Capabilities

Terminal capabilities each have a two-letter code, and are of three types:

*boolean* These indicate particular features of the terminal. For instance, an entry for a terminal that has automatic margins (an automatic RETURN and LINEFEED when the end of a line is reached) would contain a field with the boolean capability **am**.

*numeric* These give the size of the display of some other attribute. Numeric capabilities are followed by the character '**#**', and a number. An entry for a teminal with an 80-column display would have a field containing **co#80**.

*string* These indicate the character sequences used to perform particular terminal operations. String-valued capabilities, such as **ce** (clear-to-end-of-line sequence) are given by the two-letter code, followed by the character '**=**', and a string (which ends at the following **:** field delimiter).

A delay factor, in milliseconds may appear after the '**=**'. Padding characters are supplied by **tputs** after the remainder of the string is sent. The delay can be either a number, or a number followed by the character '*****', which indicates that the proportional padding is required, in which case the number given is the

amount of padding for each line affected by an operation using that capability. (In the case of an insert-character operation, the factor is still the number of *lines* affected; this is always 1 unless the terminal has **in** and the software uses it.)

When a * is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

**Comments**

To comment-out a capability field, insert a '.' (period) as the first character in that field (following the :).

**Escape Sequence Codes**

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there:

| | |
|---|---|
| **\E** | maps to ESC |
| **^X** | maps to CTRL-*X* for any appropriate character *X* |
| **\n** | maps to LINEFEED |
| **\r** | maps to RETURN |
| **\t** | maps to TAB |
| **\b** | maps to BACKSPACE |
| **\f** | maps to FORMFEED |

Finally, characters may be given as three octal digits after a backslash (for example, **\123**), and the characters ^ (caret) and \ (backslash) may be given as \^ and \\ respectively.

If it is necessary to place a : in a capability it must be escaped in octal as **\072**.

If it is necessary to place a NUL character in a string capability it must be encoded as **\200**. (The routines that deal with **termcap** use C strings and strip the high bits of the output very late, so that a **\200** comes out as a **\000** would.)

**Parameterized Strings**

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**(3S)-like escapes (*%x*) in it; other characters are passed through unchanged. For example, to address the cursor, the **cm** capability is given, using two parameters: the row and column to move to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory. If the terminal has memory-relative cursor addressing, that can be indicated by an analogous CM capability.)

The % escapes have the following meanings:

| | |
|---|---|
| **%%** | produce the character % |
| **%d** | output *value* as in **printf %d** |
| **%2** | output *value* as in **printf %2d** |
| **%3** | output *value* as in **printf %3d** |
| **%.** | output *value* as in **printf %c** |
| *%+x* | add *x* to *value*, then do '**%.**' |
| *%>xy* | if *value* > *x* then add *y*, no output |
| **%r** | reverse order of two parameters, no output |
| **%i** | increment by one, no output |
| **%n** | exclusive-or all parameters with 0140 (Datamedia 2500) |
| **%B** | BCD (16*(*value*/10)) + (*value*%10), no output |
| **%D** | Reverse coding (*value* – 2*(*value*%16)), no output (Delta Data) |

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note: the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its **cm** capability is '**:cm=6\E&%r%2c%2Y:**'. Terminals that use '**%.**' need to be able to backspace the cursor (**le**) and to move the cursor up one line on the screen (**up**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (Programs using **termcap** must set terminal modes so that TAB characters are not

expanded, making \t safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM–3a, which offsets row and column by a blank character, thus it requires '**:cm=\E=%+ %+:**'.

Row or column absolute cursor addressing can be given as single-parameter capabilities **ch** (horizontal position absolute) and **cv** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cm**. If there are parameterized local motions (for example, move *n* positions to the right) these can be given as **DO, LE, RI,** and **UP** with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have **cm**, such as the Tektronix 4025.

**Delays**

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the **tset (1)** program to set terminal driver modes appropriately. Delays embedded in the capabilities **cr, sf, le, ff,** and **ta** will set the appropriate delay bits in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**. For 4.2BSD **tset**, the delays are given as numeric capabilities **dC, dN, dB, dF,** and **dT** instead.

**Similar Terminals**

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last*, and the combined length of the entries must not exceed 1024. The capabilities given before **tc** override those in the terminal type invoked by **tc**. A capability can be canceled by placing **xx@** to the left of the **tc** invocation, where *xx* is the capability. For example, the entry

**hn|2621–nl:ks@:ke@:tc=2621:**

defines a **2621–nl** that does not have the **ks** or **ke** capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

**CAPABILITIES**

The characters in the *Notes* field in the next table have the following meanings (more than one may apply to a capability):

- **N** indicates numeric parameter(s)
- **P** indicates that padding may be specified
- ***** indicates that padding may be based on the number of lines affected
- **o** indicates capability is obsolete

Obsolete capabilities have no **terminfo** equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them.

| *Name* | *Type* | *Notes* | *Description* |
|---|---|---|---|
| **!1** | *str* | | sent by shifted save key |
| **!2** | *str* | | sent by shifted suspend key |
| **!3** | *str* | | sent by shifted undo key |
| **#1** | *str* | | sent by shifted help key |
| **#2** | *str* | | sent by shifted home key |
| **#3** | *str* | | sent by shifted input key |
| **#4** | *str* | | sent by shifted left-arrow key |
| **%0** | *str* | | sent by redo key |
| **%1** | *str* | | sent by help key |
| **%2** | *str* | | sent by mark key |
| **%3** | *str* | | sent by message key |
| **%4** | *str* | | sent by move key |
| **%5** | *str* | | sent by next-object key |
| **%6** | *str* | | sent by open key |
| **%7** | *str* | | sent by options key |

| | | | |
|---|---|---|---|
| **%8** | *str* | | sent by previous-object key |
| **%9** | *str* | | sent by print or copy key |
| **%a** | *str* | | sent by shifted message key |
| **%b** | *str* | | sent by shifted move key |
| **%c** | *str* | | sent by shifted next-object key |
| **%d** | *str* | | sent by shifted options key |
| **%e** | *str* | | sent by shifted previous-object key |
| **%f** | *str* | | sent by shifted print or copy key |
| **%g** | *str* | | sent by shifted redo key |
| **%h** | *str* | | sent by shifted replace key |
| **%i** | *str* | | sent by shifted right-arrow key |
| **%j** | *str* | | sent by shifted resume key |
| **&0** | *str* | | sent by shifted cancel key |
| **&1** | *str* | | sent by ref(erence) key |
| **&2** | *str* | | sent by refresh key |
| **&3** | *str* | | sent by replace key |
| **&4** | *str* | | sent by restart key |
| **&5** | *str* | | sent by resume key |
| **&6** | *str* | | sent by save key |
| **&7** | *str* | | sent by suspend key |
| **&8** | *str* | | sent by undo key |
| **&9** | *str* | | sent by shifted beg(inning) key |
| ***0** | *str* | | sent by shifted find key |
| ***1** | *str* | | sent by shifted cmd (command) key |
| ***2** | *str* | | sent by shifted copy key |
| ***3** | *str* | | sent by shifted create key |
| ***4** | *str* | | sent by shifted delete-char key |
| ***5** | *str* | | sent by shifted delete-line key |
| ***6** | *str* | | sent by select key |
| ***7** | *str* | | sent by shifted end key |
| ***8** | *str* | | sent by shifted clear-line key |
| ***9** | *str* | | sent by shifted exit key |
| **5i** | *bool* | | printer will not echo on screen |
| **@0** | *str* | | sent by find key |
| **@1** | *str* | | sent by beg(inning) key |
| **@2** | *str* | | sent by cancel key |
| **@3** | *str* | | sent by close key |
| **@4** | *str* | | sent by cmd (command) key |
| **@5** | *str* | | sent by copy key |
| **@6** | *str* | | sent by create key |
| **@7** | *str* | | sent by end key |
| **@8** | *str* | | sent by enter/send key (unreliable) |
| **@9** | *str* | | sent by exit key |
| **AL** | *str* | *(NP*)* | add *n* new blank lines |
| **CC** | *str* | | terminal settable command character in prototype |
| **CM** | *str* | *(NP)* | memory-relative cursor motion to row *m*, column *n* |
| **DC** | *str* | *(NP*)* | delete *n* characters |
| **DL** | *str* | *(NP*)* | delete *n* lines |
| **DO** | *str* | *(NP*)* | move cursor down *n* lines |
| **EP** | *bool* | *(o)* | even parity |
| **F1-F9** | *str* | | sent by function keys 11-19 |
| **FA-FZ** | *str* | | sent by function keys 20-45 |
| **Fa-Fr** | *str* | | sent by function keys 46-63 |
| **HC** | *bool* | | cursor is hard to see |
| **HD** | *bool* | *(o)* | half-duplex |
| **IC** | *str* | *(NP*)* | insert *n* blank characters |
| **K1** | *str* | | sent by keypad upper left |
| **K2** | *str* | | sent by keypad center |

| | | | |
|---|---|---|---|
| **K3** | *str* | | sent by keypad upper right |
| **K4** | *str* | | sent by keypad lower left |
| **K5** | *str* | | sent by keypad lower right |
| **LC** | *bool* | *(o)* | lower-case only |
| **LE** | *str* | *(NP)* | move cursor left *n* positions |
| **LF** | *str* | *(P)* | turn off soft labels |
| **LO** | *str* | *(P)* | turn on soft labels |
| **MC** | *str* | *(P)* | clear left and right soft margins |
| **ML** | *str* | *(P)* | set soft left margin |
| **MR** | *str* | *(P)* | set soft right margin |
| **NL** | *bool* | *(o)* | \n is NEWLINE, not LINEFEED |
| **NP** | *bool* | | pad character does not exist |
| **NR** | *bool* | | **ti** does not reverse **te** |
| **Nl** | *num* | | number of labels on screen (start at 1) |
| **OP** | *bool* | *(o)* | odd parity |
| **RA** | *str* | *(P)* | turn off automatic margins |
| **RF** | *str* | | send next input character (for ptys) |
| **RI** | *str* | *(NP)* | move cursor right *n* positions |
| **RX** | *str* | *(P)* | turn off xoff/xon handshaking |
| **SA** | *str* | *(P)* | turn on automatic margins |
| **SF** | *str* | *(NP*)* | scroll forward *n* lines |
| **SR** | *str* | *(NP*)* | scroll backward *n* lines |
| **SX** | *str* | *(P)* | turn on xoff/xon handshaking |
| **UC** | *bool* | *(o)* | upper-case only |
| **UP** | *str* | *(NP*)* | move cursor up *n* lines |
| **XF** | *str* | | x-off character (default DC3) |
| **XN** | *str* | | x-on character (default DC1) |
| **ac** | *str* | | graphic character set pairs aAbBcC – def=VT100 |
| **ae** | *str* | *(P)* | end alternate character set |
| **al** | *str* | *(P*)* | add new blank line |
| **am** | *bool* | | terminal has automatic margins |
| **as** | *str* | *(P)* | start alternate character set |
| **bc** | *str* | *(o)* | backspace if not ^H |
| **bl** | *str* | *(P)* | audible signal (bell) |
| **bs** | *bool* | *(o)* | terminal can backspace with ^H |
| **bt** | *str* | *(P)* | back-tab |
| **bw** | *bool* | | **le** (backspace) wraps from column 0 to last column |
| **cb** | *str* | *(P)* | clear to beginning of line, inclusive |
| **cd** | *str* | *(P*)* | clear to end of display |
| **ce** | *str* | *(P)* | clear to end of line |
| **ch** | *str* | *(NP)* | set cursor column (horizontal position) |
| **cl** | *str* | *(P*)* | clear screen and home cursor |
| **cm** | *str* | *(NP)* | screen-relative cursor motion to row *m*, column *n* |
| **co** | *num* | | number of columns in a line |
| **cr** | *str* | *(P*)* | RETURN |
| **cs** | *str* | *(NP)* | change scrolling region to lines *m* through *n* (VT100) |
| **ct** | *str* | *(P)* | clear all tab stops |
| **cv** | *str* | *(NP)* | set cursor row (vertical position) |
| **dB** | *num* | *(o)* | milliseconds of **bs** delay needed (default 0) |
| **dC** | *num* | *(o)* | milliseconds of **cr** delay needed (default 0) |
| **dF** | *num* | *(o)* | milliseconds of **ff** delay needed (default 0) |
| **dN** | *num* | *(o)* | milliseconds of **nl** delay needed (default 0) |
| **dT** | *num* | *(o)* | milliseconds of horizontal tab delay needed (default 0) |
| **dV** | *num* | *(o)* | milliseconds of vertical tab delay needed (default 0) |
| **da** | *bool* | | display may be retained above the screen |
| **db** | *bool* | | display may be retained below the screen |
| **dc** | *str* | *(P*)* | delete character |
| **dl** | *str* | *(P*)* | delete line |

| | | | |
|---|---|---|---|
| **dm** | *str* | | enter delete mode |
| **do** | *str* | | down one line |
| **ds** | *str* | | disable status line |
| **eA** | *str* | *(P)* | enable graphic character set |
| **ec** | *str* | *(NP)* | erase *n* characters |
| **ed** | *str* | | end delete mode |
| **ei** | *str* | | end insert mode |
| **eo** | *bool* | | can erase overstrikes with a blank |
| **es** | *bool* | | escape can be used on the status line |
| **ff** | *str* | *(P*)* | hardcopy terminal page eject |
| **fs** | *str* | | return from status line |
| **gn** | *bool* | | generic line type (for example dialup, switch) |
| **hc** | *bool* | | hardcopy terminal |
| **hd** | *str* | | half-line down (forward 1/2 linefeed) |
| **ho** | *str* | *(P)* | home cursor |
| **hs** | *bool* | | has extra "status line" |
| **hu** | *str* | | half-line up (reverse 1/2 linefeed) |
| **hz** | *bool* | | cannot print ~s (Hazeltine) |
| **i1** | *str* | | terminal initialization string (**terminfo** only) |
| **i3** | *str* | | terminal initialization string (**terminfo** only) |
| **iP** | *str* | | pathname of program for initialization (**terminfo** only) |
| **ic** | *str* | *(P*)* | insert character |
| **if** | *str* | | name of file containing initialization string |
| **im** | *str* | | enter insert mode |
| **in** | *bool* | | insert mode distinguishes nulls |
| **ip** | *str* | *(P*)* | insert pad after character inserted |
| **is** | *str* | | terminal initialization string |
| **it** | *num* | | tab stops initially every *n* positions |
| **k0-k9** | *str* | | sent by function keys 0-9 |
| **k;** | *str* | | sent by function key 10 |
| **kA** | *str* | | sent by insert-line key |
| **kB** | *str* | | sent by back-tab key |
| **kC** | *str* | | sent by clear-screen or erase key |
| **kD** | *str* | | sent by delete-character key |
| **kE** | *str* | | sent by clear-to-end-of-line key |
| **kF** | *str* | | sent by scroll-forward/down key |
| **kH** | *str* | | sent by home-down key |
| **kI** | *str* | | sent by insert-character or enter-insert-mode key |
| **kL** | *str* | | sent by delete-line key |
| **kM** | *str* | | sent by insert key while in insert mode |
| **kN** | *str* | | sent by next-page key |
| **kP** | *str* | | sent by previous-page key |
| **kR** | *str* | | sent by scroll-backward/up key |
| **kS** | *str* | | sent by clear-to-end-of-screen key |
| **kT** | *str* | | sent by set-tab key |
| **ka** | *str* | | sent by clear-all-tabs key |
| **kb** | *str* | | sent by backspace key |
| **kd** | *str* | | sent by down-arrow key |
| **ke** | *str* | | out of "keypad transmit" mode |
| **kh** | *str* | | sent by home key |
| **kl** | *str* | | sent by left-arrow key |
| **km** | *bool* | | has a "meta" key (shift, sets parity bit) |
| **kn** | *num* | *(o)* | number of function (**k0**–**k9**) keys (default 0) |
| **ko** | *str* | *(o)* | termcap entries for other non-function keys |
| **kr** | *str* | | sent by right-arrow key |
| **ks** | *str* | | put terminal in "keypad transmit" mode |
| **kt** | *str* | | sent by clear-tab key |
| **ku** | *str* | | sent by up-arrow key |

| | | | |
|---|---|---|---|
| **l0-l9** | *str* | | labels on function keys 0-9 if not f0-f9 |
| **la** | *str* | | label on function key 10 if not f10 |
| **le** | *str* | *(P)* | move cursor left one position |
| **lh** | *num* | | number of rows in each label |
| **li** | *num* | | number of lines on screen or page |
| **ll** | *str* | | last line, first column |
| **lm** | *num* | | lines of memory if > **li** (0 means varies) |
| **lw** | *num* | | number of columns in each label |
| **ma** | *str* | *(o)* | arrow key map (used by *vi* version 2 only) |
| **mb** | *str* | | turn on blinking attribute |
| **md** | *str* | | turn on bold (extra bright) attribute |
| **me** | *str* | | turn off all attributes |
| **mh** | *str* | | turn on half-bright attribute |
| **mi** | *bool* | | safe to move while in insert mode |
| **mk** | *str* | | turn on blank attribute (characters invisible) |
| **ml** | *str* | *(o)* | memory lock on above cursor |
| **mm** | *str* | | turn on "meta mode" (8th bit) |
| **mo** | *str* | | turn off "meta mode" |
| **mp** | *str* | | turn on protected attribute |
| **mr** | *str* | | turn on reverse-video attribute |
| **ms** | *bool* | | safe to move in standout modes |
| **mu** | *str* | *(o)* | memory unlock (turn off memory lock) |
| **nc** | *bool* | *(o)* | no correctly-working **cr** (Datamedia 2500, Hazeltine 2000) |
| **nd** | *str* | | non-destructive space (cursor right) |
| **nl** | *str* | *(o)* | NEWLINE character if not |
| **ns** | *bool* | *(o)* | terminal is a CRT but does not scroll |
| **nw** | *str* | *(P)* | NEWLINE (behaves like **cr** followed by **do**) |
| **nx** | *bool* | | padding will not work, xoff/xon required |
| **os** | *bool* | | terminal overstrikes |
| **pO** | *str* | *(N)* | turn on the printer for *n* bytes |
| **pb** | *num* | | lowest baud where delays are required |
| **pc** | *str* | | pad character (default NUL) |
| **pf** | *str* | | turn off the printer |
| **pk** | *str* | | program function key *n* to type string *s* (**terminfo** only) |
| **pl** | *str* | | program function key *n* to execute string *s* (**terminfo** only) |
| **pn** | *str* | *(NP)* | program label *n* to show string *s* (**terminfo** only) |
| **po** | *str* | | turn on the printer |
| **ps** | *str* | | print contents of the screen |
| **pt** | *bool* | *(o)* | has hardware tab stops (may need to be set with **is**) |
| **px** | *str* | | program function key *n* to transmit string *s* (**terminfo** only) |
| **r1** | *str* | | reset terminal completely to sane modes (**terminfo** only) |
| **r2** | *str* | | reset terminal completely to sane modes (**terminfo** only) |
| **r3** | *str* | | reset terminal completely to sane modes (**terminfo** only) |
| **rP** | *str* | *(P)* | like **ip** but when in replace mode |
| **rc** | *str* | *(P)* | restore cursor to position of last **sc** |
| **rf** | *str* | | name of file containing reset string |
| **ri** | *?* | | unkown at present |
| **rp** | *str* | *(NP*)* | repeat character *c* *n* times |
| **rs** | *str* | | reset terminal completely to sane modes |
| **sa** | *str* | *(NP)* | define the video attributes (9 parameters) |
| **sc** | *str* | *(P)* | save cursor position |
| **se** | *str* | | end standout mode |
| **sf** | *str* | *(P)* | scroll text up |
| **sg** | *num* | | number of garbage chars left by **so** or **se** (default 0) |
| **so** | *str* | | begin standout mode |
| **sr** | *str* | *(P)* | scroll text down |
| **st** | *str* | | set a tab stop in all rows, current column |
| **ta** | *str* | *(P)* | move cursor to next 8-position hardware tab stop |

| | | | |
|---|---|---|---|
| **tc** | *str* | | entry of similar terminal – must be last |
| **te** | *str* | | string to end programs that use **termcap** |
| **ti** | *str* | | string to begin programs that use **termcap** |
| **ts** | *str* | *(N)* | go to status line, column *n* |
| **uc** | *str* | | underscore one character and move past it |
| **ue** | *str* | | end underscore mode |
| **ug** | *num* | | number of garbage chars left by **us** or **ue** (default 0) |
| **ul** | *bool* | | underline character overstrikes |
| **up** | *str* | | upline (cursor up) |
| **us** | *str* | | start underscore mode |
| **vb** | *str* | | visible bell (must not move cursor) |
| **ve** | *str* | | make cursor appear normal (undo **vs**/**vi**) |
| **vi** | *str* | | make cursor invisible |
| **vs** | *str* | | make cursor very visible |
| **vt** | *num* | | virtual terminal number (not supported on all systems) |
| **wi** | *str* | *(N)* | set current window to lines *i* through *j*, columns *m* through *n* |
| **ws** | *num* | | number of columns in status line |
| **xb** | *bool* | | Beehive (f1=ESC, f2=^C) |
| **xn** | *bool* | | NEWLINE ignored after 80 cols (Concept) |
| **xo** | *bool* | | terminal uses xoff/xon handshaking |
| **xr** | *bool* | *(o)* | RETURN acts like **ce cr nl** (Delta Data) |
| **xs** | *bool* | | standout not erased by overwriting (Hewlett-Packard) |
| **xt** | *bool* | | TAB characters destructive, magic **so** char (Teleray 1061) |
| **xx** | *bool* | *(o)* | Tektronix 4025 insert-line |

## ENVIRONMENT

If the environment variable **TERMCAP** contains an absolute pathname, programs look to that file for terminal descriptions, rather than **/usr/share/lib/termcap**. If the value of this varible is in the form of a **termcap** entry, programs use that value for the terminal description.

## FILES

**/usr/share/lib/termcap** file containing terminal descriptions

## SEE ALSO

**ex**(1), **more**(1), **tset**(1), **ul**(1), **vi**(1), **curses**(3X), **printf**(3S), **termcap**(3X), **term**(5V), **terminfo**(5V)

*System and Network Administration*

## CAVEATS AND BUGS

UNIX System V uses **terminfo**(5V) rather than **termcap**. SunOS supports either **termcap** or **terminfo**(5V) terminal databases, depending on whether you link with the **termcap**(3X) or **curses**(3V) libraries. Transitions between the two should be relatively painless if capabilities flagged as "obsolete" are avoided.

**vi** allows only 256 characters for string capabilities, and the routines in **termcap**(3X) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped NEWLINE characters) may not exceed 1024.

Not all programs support all entries.

NAME

terminfo – terminal capability data base

SYNOPSIS

**/usr/share/lib/terminfo/?/***

AVAILABILITY

This database is available with the *System V* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**terminfo** is a compiled database (see **tic**(8V)) describing the capabilities of terminals. Terminals are described in **terminfo** source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, and by libraries such as **curses**(3V), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the –I option of **infocmp**(8V).

Entries in **terminfo** source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the **terminfo** database gives the name by which **terminfo** knows the terminal, separated by pipe (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to which the environment variable TERM would normally be set), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions:

- The particular piece of hardware making up the terminal should have a root name chosen; for example, for the Hewlett-Packard 2621, **hp2621**. This name should not contain hyphens.
- Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, a **vt100** in 132-column mode would be given as: **vt100–w**. The following suffixes should be used where possible:

| *Suffix* | *Meaning* | *Example* |
|---|---|---|
| **–w** | wide mode (more than 80 columns) | **vt100–w** |
| **–am** | with automatic margins (usually default) | **vt100–am** |
| **–nam** | without automatic margins | **vt100–nam** |
| *–n* | number of lines on the screen | **aaa–60** |
| **–na** | no arrow keys (leave them in local) | **concept100–na** |
| *–n***p** | number of pages of memory | **concept100–4p** |
| **–rv** | reverse video | **concept100–rv** |

CAPABILITIES

In the table below, the **Variable** is the name by which the C programmer (at the **terminfo** level) accesses the capability. The **capname** is the short name for this variable used in the text of the database. It is used by a person updating the database and by the **tput**(1V) command when asking what the value of the capability is for a particular terminal. The **Termcap Code** is a two-letter code that corresponds to the old **termcap** capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the table below, have names beginning with '**key_**'. The following indicators may appear at the end of the **Description** for a variable.

(G) indicates that the string is passed through **tparm( )** with parameters (parms) as given ($\#_i$).
(*) indicates that padding may be based on the number of lines affected.
($\#_i$) indicates the $i^{th}$ parameter.

| *Variable* | *Capname* | *Termcap* | *Description* |
|---|---|---|---|
| *Boolean* | | | |
| **auto_left_margin** | **bw** | **bw** | **cub1** wraps from column 0 to last column |
| **auto_right_margin** | **am** | **am** | Terminal has automatic margins |
| **no_esc_ctlc** | **xsb** | **xb** | Beehive (f1=ESC, f2=^C) |
| **ceol_standout_glitch** | **xhp** | **xs** | Standout not erased by overwriting (Hewlett-Packard) |
| **eat_newline_glitch** | **xenl** | **xn** | NEWLINE ignored after 80 cols (Concept) |
| **erase_overstrike** | **eo** | **eo** | Can erase overstrikes with a blank |
| **generic_type** | **gn** | **gn** | Generic line type (for example, dialup, switch). |
| **hard_copy** | **hc** | **hc** | Hardcopy terminal |
| **hard_cursor** | **chts** | **HC** | Cursor is hard to see. |
| **has_meta_key** | **km** | **km** | Has a meta key (shift, sets parity bit) |
| **has_status_line** | **hs** | **hs** | Has extra "status line" |
| **insert_null_glitch** | **in** | **in** | Insert mode distinguishes nulls |
| **memory_above** | **da** | **da** | Display may be retained above the screen |
| **memory_below** | **db** | **db** | Display may be retained below the screen |
| **move_insert_mode** | **mir** | **mi** | Safe to move while in insert mode |
| **move_standout_mode** | **msgr** | **ms** | Safe to move in standout modes |
| **needs_xon_xoff** | **nxon** | **nx** | Padding will not work, xon/xoff required |
| **non_rev_rmcup** | **nrrmc** | **NR** | **smcup** does not reverse **rmcup** |
| **no_pad_char** | **npc** | **NP** | Pad character does not exist |
| **over_strike** | **os** | **os** | Terminal overstrikes on hard-copy terminal |
| **prtr_silent** | **mc5i** | **5i** | Printer will not echo on screen. |
| **status_line_esc_ok** | **eslok** | **es** | Escape can be used on the status line |
| **dest_tabs_magic_smso** | **xt** | **xt** | Destructive TAB characters, magic **smso** char (Teleray 1061) |
| **tilde_glitch** | **hz** | **hz** | Hazeltine; cannot print tildes(~) |
| **transparent_underline** | **ul** | **ul** | Underline character overstrikes |
| **xon_xoff** | **xon** | **xo** | Terminal uses xon/xoff handshaking |
| *Number* | | | |
| **columns** | **cols** | **co** | Number of columns in a line |
| **init_tabs** | **it** | **it** | tab stops initially every # spaces. |
| **label_height** | **lh** | **lh** | Number of rows in each label |
| **label_width** | **lw** | **lw** | Number of cols in each label |
| **lines** | **lines** | **li** | Number of lines on screen or page |
| **lines_of_memory** | **lm** | **lm** | Lines of memory if > **lines**; **0** means varies |
| **magic_cookie_glitch** | **xmc** | **sg** | Number blank chars left by **smso** or **rmso** |
| **num_labels** | **nlab** | **Nl** | Number of labels on screen (start at 1) |
| **padding_baud_rate** | **pb** | **pb** | Lowest baud rate where padding needed |
| **virtual_terminal** | **vt** | **vt** | Virtual terminal number (not supported on all systems) |
| **width_status_line** | **wsl** | **ws** | Number of columns in status line |
| *String* | | | |
| **acs_chars** | **acsc** | **ac** | Graphic charset pairs aAbBcC - def=VT100 |
| **back_tab** | **cbt** | **bt** | Back tab |
| **bell** | **bel** | **bl** | Audible signal (bell) |
| **carriage_return** | **cr** | **cr** | RETURN (*) |
| **change_scroll_region** | **csr** | **cs** | Change to lines #1 through #2 (VT100) (G) |

| | | | |
|---|---|---|---|
| **char_padding** | **rmp** | **rP** | Like **ip** but when in replace mode |
| **clear_all_tabs** | **tbc** | **ct** | Clear all tab stops |
| **clear_margins** | **mgc** | **MC** | Clear left and right soft margins |
| **clear_screen** | **clear** | **cl** | Clear screen and home cursor (*) |
| **clr_bol** | **el1** | **cb** | Clear to beginning of line, inclusive |
| **clr_eol** | **el** | **ce** | Clear to end of line |
| **clr_eos** | **ed** | **cd** | Clear to end of display (*) |
| **column_address** | **hpa** | **ch** | Horizontal position absolute (G) |
| **command_character** | **cmdch** | **CC** | Terminal settable command char in prototype |
| **cursor_address** | **cup** | **cm** | Cursor motion to row #1 col #2 (G) |
| **cursor_down** | **cud1** | **do** | Down one line |
| **cursor_home** | **home** | **ho** | Home cursor (if no **cup**) |
| **cursor_invisible** | **civis** | **vi** | Make cursor invisible |
| **cursor_left** | **cub1** | **le** | Move cursor left one SPACE. |
| **cursor_mem_address** | **mrcup** | **CM** | Memory relative cursor addressing (G) |
| **cursor_normal** | **cnorm** | **ve** | Make cursor appear normal (undo **cvvis/civis**) |
| **cursor_right** | **cuf1** | **nd** | Non-destructive space (cursor right) |
| **cursor_to_ll** | **ll** | **ll** | Last line, first column (if no **cup**) |
| **cursor_up** | **cuu1** | **up** | Upline (cursor up) |
| **cursor_visible** | **cvvis** | **vs** | Make cursor very visible |
| **delete_character** | **dch1** | **dc** | Delete character (*) |
| **delete_line** | **dl1** | **dl** | Delete line (*) |
| **dis_status_line** | **dsl** | **ds** | Disable status line |
| **down_half_line** | **hd** | **hd** | Half-line down (forward 1/2 LINEFEED) |
| **ena_acs** | **enacs** | **eA** | Enable alternate char set |
| **enter_alt_charset_mode** | **smacs** | **as** | Start alternate character set |
| **enter_am_mode** | **smam** | **SA** | Turn on automatic margins |
| **enter_blink_mode** | **blink** | **mb** | Turn on blinking |
| **enter_bold_mode** | **bold** | **md** | Turn on bold (extra bright) mode |
| **enter_ca_mode** | **smcup** | **ti** | String to begin programs that use **cup** |
| **enter_delete_mode** | **smdc** | **dm** | Delete mode (enter) |
| **enter_dim_mode** | **dim** | **mh** | Turn on half-bright mode |
| **enter_insert_mode** | **smir** | **im** | Insert mode (enter); |
| **enter_protected_mode** | **prot** | **mp** | Turn on protected mode |
| **enter_reverse_mode** | **rev** | **mr** | Turn on reverse video mode |
| **enter_secure_mode** | **invis** | **mk** | Turn on blank mode (chars invisible) |
| **enter_standout_mode** | **smso** | **so** | Begin standout mode |
| **enter_underline_mode** | **smul** | **us** | Start underscore mode |
| **enter_xon_mode** | **smxon** | **SX** | Turn on xon/xoff handshaking |
| **erase_chars** | **ech** | **ec** | Erase #1 characters (G) |
| **exit_alt_charset_mode** | **rmacs** | **ae** | End alternate character set |
| **exit_am_mode** | **rmam** | **RA** | Turn off automatic margins |
| **exit_attribute_mode** | **sgr0** | **me** | Turn off all attributes |
| **exit_ca_mode** | **rmcup** | **te** | String to end programs that use **cup** |
| **exit_delete_mode** | **rmdc** | **ed** | End delete mode |
| **exit_insert_mode** | **rmir** | **ei** | End insert mode; |
| **exit_standout_mode** | **rmso** | **se** | End standout mode |
| **exit_underline_mode** | **rmul** | **ue** | End underscore mode |
| **exit_xon_mode** | **rmxon** | **RX** | Turn off xon/xoff handshaking |
| **flash_screen** | **flash** | **vb** | Visible bell (must not move cursor) |
| **form_feed** | **ff** | **ff** | Hardcopy terminal page eject (*) |
| **from_status_line** | **fsl** | **fs** | Return from status line |
| **init_1string** | **is1** | **i1** | Terminal initialization string |
| **init_2string** | **is2** | **is** | Terminal initialization string |
| **init_3string** | **is3** | **i3** | Terminal initialization string |
| **init_file** | **if** | **if** | Name of initialization file containing **is** |
| **init_prog** | **iprog** | **iP** | Path name of program for init. |
| **insert_character** | **ich1** | **ic** | Insert character |

| | | | |
|---|---|---|---|
| **insert_line** | **il1** | **al** | Add new blank line (*) |
| **insert_padding** | **ip** | **ip** | Insert pad after character inserted (*) |
| **key_a1** | **ka1** | **K1** | KEY_A1, 0534, Upper left of keypad |
| **key_a3** | **ka3** | **K3** | KEY_A3, 0535, Upper right of keypad |
| **key_b2** | **kb2** | **K2** | KEY_B2, 0536, Center of keypad |
| **key_backspace** | **kbs** | **kb** | KEY_BACKSPACE, 0407, Sent by BACKSPACE key |
| **key_beg** | **kbeg** | **@1** | KEY_BEG, 0542, Sent by beg(inning) key |
| **key_btab** | **kcbt** | **kB** | KEY_BTAB, 0541, Sent by back-tab key |
| **key_c1** | **kc1** | **K4** | KEY_C1, 0537, Lower left of keypad |
| **key_c3** | **kc3** | **K5** | KEY_C3, 0540, Lower right of keypad |
| **key_cancel** | **kcan** | **@2** | KEY_CANCEL, 0543, Sent by cancel key |
| **key_catab** | **ktbc** | **ka** | KEY_CATAB, 0526, Sent by clear-all-tabs key |
| **key_clear** | **kclr** | **kC** | KEY_CLEAR, 0515, Sent by clear-screen or erase key |
| **key_close** | **kclo** | **@3** | KEY_CLOSE, 0544, Sent by close key |
| **key_command** | **kcmd** | **@4** | KEY_COMMAND, 0545, Sent by cmd (command) key |
| **key_copy** | **kcpy** | **@5** | KEY_COPY, 0546, Sent by copy key |
| **key_create** | **kcrt** | **@6** | KEY_CREATE, 0547, Sent by create key |
| **key_ctab** | **kctab** | **kt** | KEY_CTAB, 0525, Sent by clear-tab key |
| **key_dc** | **kdch1** | **kD** | KEY_DC, 0512, Sent by delete-character key |
| **key_dl** | **kdl1** | **kL** | KEY_DL, 0510, Sent by delete-line key |
| **key_down** | **kcud1** | **kd** | KEY_DOWN, 0402, Sent by terminal down-arrow key |
| **key_eic** | **krmir** | **kM** | KEY_EIC, 0514, Sent by **rmir** or **smir** in insert mode |
| **key_end** | **kend** | **@7** | KEY_END, 0550, Sent by end key |
| **key_enter** | **kent** | **@8** | KEY_ENTER, 0527, Sent by enter/send key |
| **key_eol** | **kel** | **kE** | KEY_EOL, 0517, Sent by clear-to-end-of-line key |
| **key_eos** | **ked** | **kS** | KEY_EOS, 0516, Sent by clear-to-end-of-screen key |
| **key_exit** | **kext** | **@9** | KEY_EXIT, 0551, Sent by exit key |
| **key_f0** | **kf0** | **k0** | KEY_F(0), 0410, Sent by function key f0 |
| **key_f1** | **kf1** | **k1** | KEY_F(1), 0411, Sent by function key f1 |
| **key_f2** | **kf2** | **k2** | KEY_F(2), 0412, Sent by function key f2 |
| **key_f3** | **kf3** | **k3** | KEY_F(3), 0413, Sent by function key f3 |
| **key_f4** | **kf4** | **k4** | KEY_F(4), 0414, Sent by function key f4 |
| **key_f5** | **kf5** | **k5** | KEY_F(5), 0415, Sent by function key f5 |
| **key_f6** | **kf6** | **k6** | KEY_F(6), 0416, Sent by function key f6 |
| **key_f7** | **kf7** | **k7** | KEY_F(7), 0417, Sent by function key f7 |
| **key_f8** | **kf8** | **k8** | KEY_F(8), 0420, Sent by function key f8 |
| **key_f9** | **kf9** | **k9** | KEY_F(9), 0421, Sent by function key f9 |
| **key_f10** | **kf10** | **k;** | KEY_F(10), 0422, Sent by function key f10 |
| **key_f11** | **kf11** | **F1** | KEY_F(11), 0423, Sent by function key f11 |
| **key_f12** | **kf12** | **F2** | KEY_F(12), 0424, Sent by function key f12 |
| **key_f13** | **kf13** | **F3** | KEY_F(13), 0425, Sent by function key f13 |
| **key_f14** | **kf14** | **F4** | KEY_F(14), 0426, Sent by function key f14 |
| **key_f15** | **kf15** | **F5** | KEY_F(15), 0427, Sent by function key f15 |
| **key_f16** | **kf16** | **F6** | KEY_F(16), 0430, Sent by function key f16 |
| **key_f17** | **kf17** | **F7** | KEY_F(17), 0431, Sent by function key f17 |
| **key_f18** | **kf18** | **F8** | KEY_F(18), 0432, Sent by function key f18 |
| **key_f19** | **kf19** | **F9** | KEY_F(19), 0433, Sent by function key f19 |
| **key_f20** | **kf20** | **FA** | KEY_F(20), 0434, Sent by function key f20 |
| **key_f21** | **kf21** | **FB** | KEY_F(21), 0435, Sent by function key f21 |
| **key_f22** | **kf22** | **FC** | KEY_F(22), 0436, Sent by function key f22 |
| **key_f23** | **kf23** | **FD** | KEY_F(23), 0437, Sent by function key f23 |
| **key_f24** | **kf24** | **FE** | KEY_F(24), 0440, Sent by function key f24 |
| **key_f25** | **kf25** | **FF** | KEY_F(25), 0441, Sent by function key f25 |
| **key_f26** | **kf26** | **FG** | KEY_F(26), 0442, Sent by function key f26 |
| **key_f27** | **kf27** | **FH** | KEY_F(27), 0443, Sent by function key f27 |
| **key_f28** | **kf28** | **FI** | KEY_F(28), 0444, Sent by function key f28 |
| **key_f29** | **kf29** | **FJ** | KEY_F(29), 0445, Sent by function key f29 |
| **key_f30** | **kf30** | **FK** | KEY_F(30), 0446, Sent by function key f30 |

| | | | |
|---|---|---|---|
| **key_f31** | **kf31** | **FL** | KEY_F(31), 0447, Sent by function key f31 |
| **key_f32** | **kf32** | **FM** | KEY_F(32), 0450, Sent by function key f32 |
| **key_f33** | **kf33** | **FN** | KEY_F(13), 0451, Sent by function key f13 |
| **key_f34** | **kf34** | **FO** | KEY_F(34), 0452, Sent by function key f34 |
| **key_f35** | **kf35** | **FP** | KEY_F(35), 0453, Sent by function key f35 |
| **key_f36** | **kf36** | **FQ** | KEY_F(36), 0454, Sent by function key f36 |
| **key_f37** | **kf37** | **FR** | KEY_F(37), 0455, Sent by function key f37 |
| **key_f38** | **kf38** | **FS** | KEY_F(38), 0456, Sent by function key f38 |
| **key_f39** | **kf39** | **FT** | KEY_F(39), 0457, Sent by function key f39 |
| **key_f40** | **kf40** | **FU** | KEY_F(40), 0460, Sent by function key f40 |
| **key_f41** | **kf41** | **FV** | KEY_F(41), 0461, Sent by function key f41 |
| **key_f42** | **kf42** | **FW** | KEY_F(42), 0462, Sent by function key f42 |
| **key_f43** | **kf43** | **FX** | KEY_F(43), 0463, Sent by function key f43 |
| **key_f44** | **kf44** | **FY** | KEY_F(44), 0464, Sent by function key f44 |
| **key_f45** | **kf45** | **FZ** | KEY_F(45), 0465, Sent by function key f45 |
| **key_f46** | **kf46** | **Fa** | KEY_F(46), 0466, Sent by function key f46 |
| **key_f47** | **kf47** | **Fb** | KEY_F(47), 0467, Sent by function key f47 |
| **key_f48** | **kf48** | **Fc** | KEY_F(48), 0470, Sent by function key f48 |
| **key_f49** | **kf49** | **Fd** | KEY_F(49), 0471, Sent by function key f49 |
| **key_f50** | **kf50** | **Fe** | KEY_F(50), 0472, Sent by function key f50 |
| **key_f51** | **kf51** | **Ff** | KEY_F(51), 0473, Sent by function key f51 |
| **key_f52** | **kf52** | **Fg** | KEY_F(52), 0474, Sent by function key f52 |
| **key_f53** | **kf53** | **Fh** | KEY_F(53), 0475, Sent by function key f53 |
| **key_f54** | **kf54** | **Fi** | KEY_F(54), 0476, Sent by function key f54 |
| **key_f55** | **kf55** | **Fj** | KEY_F(55), 0477, Sent by function key f55 |
| **key_f56** | **kf56** | **Fk** | KEY_F(56), 0500, Sent by function key f56 |
| **key_f57** | **kf57** | **Fl** | KEY_F(57), 0501, Sent by function key f57 |
| **key_f58** | **kf58** | **Fm** | KEY_F(58), 0502, Sent by function key f58 |
| **key_f59** | **kf59** | **Fn** | KEY_F(59), 0503, Sent by function key f59 |
| **key_f60** | **kf60** | **Fo** | KEY_F(60), 0504, Sent by function key f60 |
| **key_f61** | **kf61** | **Fp** | KEY_F(61), 0505, Sent by function key f61 |
| **key_f62** | **kf62** | **Fq** | KEY_F(62), 0506, Sent by function key f62 |
| **key_f63** | **kf63** | **Fr** | KEY_F(63), 0507, Sent by function key f63 |
| **key_find** | **kfnd** | **@0** | KEY_FIND, 0552, Sent by find key |
| **key_help** | **khlp** | **%1** | KEY_HELP, 0553, Sent by help key |
| **key_home** | **khome** | **kh** | KEY_HOME, 0406, Sent by home key |
| **key_ic** | **kich1** | **kI** | KEY_IC, 0513, Sent by ins-char/enter ins-mode key |
| **key_il** | **kil1** | **kA** | KEY_IL, 0511, Sent by insert-line key |
| **key_left** | **kcub1** | **kl** | KEY_LEFT, 0404, Sent by terminal left-arrow key |
| **key_ll** | **kll** | **kH** | KEY_LL, 0533, Sent by home-down key |
| **key_mark** | **kmrk** | **%2** | KEY_MARK, 0554, Sent by mark key |
| **key_message** | **kmsg** | **%3** | KEY_MESSAGE, 0555, Sent by message key |
| **key_move** | **kmov** | **%4** | KEY_MOVE, 0556, Sent by move key |
| **key_next** | **knxt** | **%5** | KEY_NEXT, 0557, Sent by next-object key |
| **key_npage** | **knp** | **kN** | KEY_NPAGE, 0522, Sent by next-page key |
| **key_open** | **kopn** | **%6** | KEY_OPEN, 0560, Sent by open key |
| **key_options** | **kopt** | **%7** | KEY_OPTIONS, 0561, Sent by options key |
| **key_ppage** | **kpp** | **kP** | KEY_PPAGE, 0523, Sent by previous-page key |
| **key_previous** | **kprv** | **%8** | KEY_PREVIOUS, 0562, Sent by previous-object key |
| **key_print** | **kprt** | **%9** | KEY_PRINT, 0532, Sent by print or copy key |
| **key_redo** | **krdo** | **%0** | KEY_REDO, 0563, Sent by redo key |
| **key_reference** | **kref** | **&1** | KEY_REFERENCE, 0564, Sent by ref(erence) key |
| **key_refresh** | **krfr** | **&2** | KEY_REFRESH, 0565, Sent by refresh key |
| **key_replace** | **krpl** | **&3** | KEY_REPLACE, 0566, Sent by replace key |
| **key_restart** | **krst** | **&4** | KEY_RESTART, 0567, Sent by restart key |
| **key_resume** | **kres** | **&5** | KEY_RESUME, 0570, Sent by resume key |
| **key_right** | **kcuf1** | **kr** | KEY_RIGHT, 0405, Sent by terminal right-arrow key |
| **key_save** | **ksav** | **&6** | KEY_SAVE, 0571, Sent by save key |

| | | | |
|---|---|---|---|
| **key_sbeg** | **kBEG** | **&9** | KEY_SBEG, 0572, Sent by shifted beginning key |
| **key_scancel** | **kCAN** | **&0** | KEY_SCANCEL, 0573, Sent by shifted cancel key |
| **key_scommand** | **kCMD** | ***1** | KEY_SCOMMAND, 0574, Sent by shifted command key |
| **key_scopy** | **kCPY** | ***2** | KEY_SCOPY, 0575, Sent by shifted copy key |
| **key_screate** | **kCRT** | ***3** | KEY_SCREATE, 0576, Sent by shifted create key |
| **key_sdc** | **kDC** | ***4** | KEY_SDC, 0577, Sent by shifted delete-char key |
| **key_sdl** | **kDL** | ***5** | KEY_SDL, 0600, Sent by shifted delete-line key |
| **key_select** | **kslt** | ***6** | KEY_SELECT, 0601, Sent by select key |
| **key_send** | **kEND** | ***7** | KEY_SEND, 0602, Sent by shifted end key |
| **key_seol** | **kEOL** | ***8** | KEY_SEOL, 0603, Sent by shifted clear-line key |
| **key_sexit** | **kEXT** | ***9** | KEY_SEXIT, 0604, Sent by shifted exit key |
| **key_sf** | **kind** | **kF** | KEY_SF, 0520, Sent by scroll-forward/down key |
| **key_sfind** | **kFND** | ***0** | KEY_SFIND, 0605, Sent by shifted find key |
| **key_shelp** | **kHLP** | **#1** | KEY_SHELP, 0606, Sent by shifted help key |
| **key_shome** | **kHOM** | **#2** | KEY_SHOME, 0607, Sent by shifted home key |
| **key_sic** | **kIC** | **#3** | KEY_SIC, 0610, Sent by shifted input key |
| **key_sleft** | **kLFT** | **#4** | KEY_SLEFT, 0611, Sent by shifted left-arrow key |
| **key_smessage** | **kMSG** | **%a** | KEY_SMESSAGE, 0612, Sent by shifted message key |
| **key_smove** | **kMOV** | **%b** | KEY_SMOVE, 0613, Sent by shifted move key |
| **key_snext** | **kNXT** | **%c** | KEY_SNEXT, 0614, Sent by shifted next key |
| **key_soptions** | **kOPT** | **%d** | KEY_SOPTIONS, 0615, Sent by shifted options key |
| **key_sprevious** | **kPRV** | **%e** | KEY_SPREVIOUS, 0616, Sent by shifted prev key |
| **key_sprint** | **kPRT** | **%f** | KEY_SPRINT, 0617, Sent by shifted print key |
| **key_sr** | **kri** | **kR** | KEY_SR, 0521, Sent by scroll-backward/up key |
| **key_sredo** | **kRDO** | **%g** | KEY_SREDO, 0620, Sent by shifted redo key |
| **key_sreplace** | **kRPL** | **%h** | KEY_SREPLACE, 0621, Sent by shifted replace key |
| **key_sright** | **kRIT** | **%i** | KEY_SRIGHT, 0622, Sent by shifted right-arrow key |
| **key_srsume** | **kRES** | **%j** | KEY_SRSUME, 0623, Sent by shifted resume key |
| **key_ssave** | **kSAV** | **!1** | KEY_SSAVE, 0624, Sent by shifted save key |
| **key_ssuspend** | **kSPD** | **!2** | KEY_SSUSPEND, 0625, Sent by shifted suspend key |
| **key_stab** | **khts** | **kT** | KEY_STAB, 0524, Sent by set-tab key |
| **key_sundo** | **kUND** | **!3** | KEY_SUNDO, 0626, Sent by shifted undo key |
| **key_suspend** | **kspd** | **&7** | KEY_SUSPEND, 0627, Sent by suspend key |
| **key_undo** | **kund** | **&8** | KEY_UNDO, 0630, Sent by undo key |
| **key_up** | **kcuu1** | **ku** | KEY_UP, 0403, Sent by terminal up-arrow key |
| **keypad_local** | **rmkx** | **ke** | Out of "keypad-transmit" mode |
| **keypad_xmit** | **smkx** | **ks** | Put terminal in "keypad-transmit" mode |
| **lab_f0** | **lf0** | **l0** | Labels on function key f0 if not f0 |
| **lab_f1** | **lf1** | **l1** | Labels on function key f1 if not f1 |
| **lab_f2** | **lf2** | **l2** | Labels on function key f2 if not f2 |
| **lab_f3** | **lf3** | **l3** | Labels on function key f3 if not f3 |
| **lab_f4** | **lf4** | **l4** | Labels on function key f4 if not f4 |
| **lab_f5** | **lf5** | **l5** | Labels on function key f5 if not f5 |
| **lab_f6** | **lf6** | **l6** | Labels on function key f6 if not f6 |
| **lab_f7** | **lf7** | **l7** | Labels on function key f7 if not f7 |
| **lab_f8** | **lf8** | **l8** | Labels on function key f8 if not f8 |
| **lab_f9** | **lf9** | **l9** | Labels on function key f9 if not f9 |
| **lab_f10** | **lf10** | **la** | Labels on function key f10 if not f10 |
| **label_off** | **rmln** | **LF** | Turn off soft labels |
| **label_on** | **smln** | **LO** | Turn on soft labels |
| **meta_off** | **rmm** | **mo** | Turn off "meta mode" |
| **meta_on** | **smm** | **mm** | Turn on "meta mode" (8th bit) |
| **newline** | **nel** | **nw** | NEWLINE (behaves like **cr** followed by **lf**) |
| **pad_char** | **pad** | **pc** | Pad character (rather than null) |
| **parm_dch** | **dch** | **DC** | Delete #1 chars (G*) |
| **parm_delete_line** | **dl** | **DL** | Delete #1 lines (G*) |
| **parm_down_cursor** | **cud** | **DO** | Move cursor down #1 lines. (G*) |
| **parm_ich** | **ich** | **IC** | Insert #1 blank chars (G*) |

| | | | |
|---|---|---|---|
| **parm_index** | **indn** | **SF** | Scroll forward #1 lines. (G) |
| **parm_insert_line** | **il** | **AL** | Add #1 new blank lines (G*) |
| **parm_left_cursor** | **cub** | **LE** | Move cursor left #1 spaces (G) |
| **parm_right_cursor** | **cuf** | **RI** | Move cursor right #1 spaces. (G*) |
| **parm_rindex** | **rin** | **SR** | Scroll backward #1 lines. (G) |
| **parm_up_cursor** | **cuu** | **UP** | Move cursor up #1 lines. (G*) |
| **pkey_key** | **pfkey** | **pk** | Prog funct key #1 to type string #2 |
| **pkey_local** | **pfloc** | **pl** | Prog funct key #1 to execute string #2 |
| **pkey_xmit** | **pfx** | **px** | Prog funct key #1 to xmit string #2 |
| **plab_norm** | **pln** | **pn** | Prog label #1 to show string #2 |
| **print_screen** | **mc0** | **ps** | Print contents of the screen |
| **prtr_non** | **mc5p** | **pO** | Turn on the printer for #1 bytes |
| **prtr_off** | **mc4** | **pf** | Turn off the printer |
| **prtr_on** | **mc5** | **po** | Turn on the printer |
| **repeat_char** | **rep** | **rp** | Repeat char #1 #2 times (G*) |
| **req_for_input** | **rfi** | **RF** | Send next input char (for ptys) |
| **reset_1string** | **rs1** | **r1** | Reset terminal completely to sane modes |
| **reset_2string** | **rs2** | **r2** | Reset terminal completely to sane modes |
| **reset_3string** | **rs3** | **r3** | Reset terminal completely to sane modes |
| **reset_file** | **rf** | **rf** | Name of file containing reset string |
| **restore_cursor** | **rc** | **rc** | Restore cursor to position of last **sc** |
| **row_address** | **vpa** | **cv** | Vertical position absolute (G) |
| **save_cursor** | **sc** | **sc** | Save cursor position. |
| **scroll_forward** | **ind** | **sf** | Scroll text up |
| **scroll_reverse** | **ri** | **sr** | Scroll text down |
| **set_attributes** | **sgr** | **sa** | Define the video attributes #1-#9 (G) |
| **set_left_margin** | **smgl** | **ML** | Set soft left margin |
| **set_right_margin** | **smgr** | **MR** | Set soft right margin |
| **set_tab** | **hts** | **st** | Set a tab stop in all rows, current column. |
| **set_window** | **wind** | **wi** | Current window is lines #1-#2 cols #3-#4 (G) |
| **tab** | **ht** | **ta** | Move the cursor to the next 8 space hardware tab stop. |
| **to_status_line** | **tsl** | **ts** | Go to status line, col #1 (G) |
| **underline_char** | **uc** | **uc** | Underscore one char and move past it |
| **up_half_line** | **hu** | **hu** | Half-line up (reverse 1/2 line-feed) |
| **xoff_character** | **xoffc** | **XF** | X-off character |
| **xon_character** | **xonc** | **XN** | X-on character |

## SAMPLE ENTRY

The following entry, which describes the Concept 100 terminal, is among the more complex entries in the **terminfo** file as of this writing.

```
concept100|c100|concept|c104|c100-4p|concept 100,
    am, db, eo, in, mir, ul, xenl, cols#80, lines#24, pb#9600, vt#8,
    bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew, cr=^M$<9>,
    cub1=^H, cud1=^J, cuf1=\E=, cup=\Ea%p1%' '%+%c%p2%' '%+%c, cuu1=\E;,
    cvvis=\EW, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
    ed=\E^C$<16*>, el=\E^U$<16>, flash=\Ek$<20>\EK, ht=\t$<8>,
    il1=\E^R$<3*>, ind=^J, .ind=^J$<9>, ip=$<16*>,
    is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\0\Eo&\0\Eo\47\E,
    kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;, kf1=\E5,
    kf2=\E6, kf3=\E7, khome=\E?, prot=\EI,
    rep=\Er%p1%c%p2%' '%+%c$<.2*>, rev=\ED,
    rmcup=\Ev\s\s\s\s$<6>\Ep\r\n, rmir=\E\0, rmkx=\Ex,
    rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\0,
    smcup=\EU\Ev\s\s8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
    smul=\EG,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Lines beginning with # are taken as comment lines. Capabilities in **terminfo** are of three types: boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or particular features, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

**Types of Capabilities**

All capabilities have names. For instance, the fact that the Concept has *automatic margins* (that is, an automatic RETURN and LINEFEED when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character # and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value **80** for the Concept. The value may be specified in decimal, octal or hexadecimal using normal C conventions.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the two- to five-character capname, an '=', and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in **$<..>** brackets, as in '**el=\EK$<3>**', and padding characters are supplied by **tputs**( ) (see **curses**(3V)) to provide this delay. The delay can be either a number, for example, **20**, or a number followed by an * (for example, **3***), a / (for example, **5/**), or both (for example, **10*/** ). A * indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of lines affected. This is always one unless the terminal has **in** and the software uses it.) When a * is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.) A / indicates that the padding is mandatory. Otherwise, if the terminal has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**.

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there:

| | |
|---|---|
| **\E, \e** | map to ESC |
| **^X** | maps to CTRL-*X* for any appropriate character *X* |
| **\n** | maps to NEWLINE |
| **\l** | maps to LINEFEED |
| **\r** | maps to RETURN |
| **\t** | maps to TAB |
| **\b** | maps to BACKSPACE |
| **\f** | maps to FORMFEED |
| **\s** | maps to SPACE |
| **\0** | maps to NUL |

(**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a backslash (for example, **\123**), and the characters ^ (caret), \ (backslash), : (colon), and , (comma) may be given as \^, \\, \:, and \,, respectively.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above. Note: capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

**Preparing Descriptions**

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **terminfo** and to build up a description gradually, using partial descriptions with some *curses*-based application to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **terminfo** file to describe it or bugs in the application. To test a new terminal description, set the environment variable **TERMINFO** to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/share/lib/terminfo**. To get the padding for insert-line correct (if the terminal manufacturer did not document it) a severe test is to insert 16 lines into the middle of a full screen at 9600 baud. If the display is corrupted, more padding is usu-

ally needed. A similar test can be used for insert-character.

**Basic Capabilities**

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal has a screen, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as Tektronix 4010 series, as well as hard-copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be RETURN, CTRL-M.) If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**. If the terminal uses the xon-xoff flow-control protocol, like most terminals, specify **xon**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over; for example, you would not normally use **cuf1**=\s because the SPACE would erase the character moved over.

A very important point here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and should never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the **terminfo** file usually assumes that this is on; that is, **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (NEWLINE). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the model 33 teletype is described as

```
33|tty33|tty|model 33 teletype,
        bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM–3 is described as

```
adm3|lsi adm3,
        am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
        ind=^J, lines#24,
```

**Parameterized Strings**

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with **printf**(3S)-like escapes (**%x**) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate it in the manner of a Reverse Polish Notation (postfix) calculator. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Binary operations are in postfix form with the operands in the usual order. That is, to get x–5 one would use '**%gx%{5}%–**'.

The % encodings have the following meanings:

| | |
|---|---|
| **%%** | outputs **%** |
| **%[[:]***flags***][***width***[.***precision***]][doxXs]** | as in **printf**(3S), flags are [–+#] and SPACE |
| **%c** | print **pop()** gives **%c** |
| **%p[1-9]** | push $i^{th}$ parm |
| **%P[a-z]** | set variable [a-z] to **pop()** |
| **%g[a-z]** | get variable [a-z] and push it |
| **%'***c***'** | push char constant *c* |
| **%{***nn***}** | push decimal constant *nn* |
| **%l** | push **strlen(pop())** |
| **%+ %– %* %/ %m** | arithmetic (**%m** is mod): **push(pop() op pop())** |
| **%& %\| %^** | bit operations: **push(pop() op pop())** |
| **%= %> %<** | logical operations: **push(pop() op pop())** |
| **%A %O** | logical operations: and, or |
| **%! %~** | unary operations: **push(op pop())** |
| **%i** | (for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present |
| **%?***expr* **%t***thenpart* **%e***elsepart***%;** | if-then-else, '**%e***elsepart*' is optional; else-if's are possible in Algol 68: **%? $c_1$ %t $b_1$ %e $c_2$ %t $b_2$ %e $c_3$ %t $b_3$ %e $c_4$ %t $b_4$ %e $b_5$%;** $c_i$ are conditions, $b_i$ are bodies. |

If the '–' flag is used with '**%[doxXs]**', then a colon (:) must be placed between the '**%**' and the '–' to differentiate the flag from the binary '**%–**' operator, for example, '**%:–16.16s**'.

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note: the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its **cup** capability is:

cup=\E&a%p2%2.2dc%p1%2.2dY$<6>

The Micro-Term ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, '**cup**=^T%p1%c%p2%c'. Terminals which use **%c** need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with **terminfo** set tty modes so that TAB characters are never expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus '**cup**=\E=%p1%'\s'%+%c%p2%'\s'%+%c'. After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

**Cursor Motions**

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note: the home

position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

If the terminal has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the Tektronix 4025.

**Area Clears**

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

**Insert/Delete Line**

If the terminal can open a new blank line before the line where the cursor is, this should be given as '**il1**'; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as '**dl1**'; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

**Insert/Delete Character**

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using **terminfo**. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type '**abc def**' using local cursor motions (not SPACE characters) between the **abc** and the **def**. Then position the cursor before the **abc** and put the termi-

nal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the **abc** shifts over to the **def** which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for "insert null". While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped blanks) we have seen no terminals whose insert mode cannot be described with the single attribute.

**terminfo** can describe both terminals which have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals which send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an "insert mode" and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a TAB character after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

**Highlighting, Underlining, and Visible Bells**

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode* (see **curses**(3V)), representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse-video plus half-bright is good, or reverse-video alone; however, different users have different preferences on different terminals.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blanks on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many blanks are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Micro-Term MIME, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking), **bold** (bold or extra-bright), **dim** (dim or half-bright), **invis** (blanking or invisible text), **prot** (protected), **rev** (reverse-video), **sgr0** (turn off all attribute modes), **smacs** (enter alternate-character-set mode), and **rmacs** (exit alternate-character-set mode). Turning on any of these modes singly may or may not turn off other modes. If a command is necessary before alternate character set mode is entered, give the sequence in **enacs** (enable alternate-character-set mode).

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of either of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the Tektronix 4025, where **smcup** sets the command character to be the one used by **terminfo**. If the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability **ul**. For terminals where a character overstriking another leaves both characters on the screen, give the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Example of highlighting: assume that the terminal under question needs the following escape sequences to turn on various modes.

| **tparm parameter** | **attribute** | **escape sequence** |
|---|---|---|
| | none | \E[0m |
| p1 | standout | \E[0;4;7m |
| p2 | underline | \E[0;3m |
| p3 | reverse | \E[0;4m |
| p4 | blink | \E[0;5m |
| p5 | dim | \E[0;7m |
| p6 | bold | \E[0;3;4m |
| p7 | invis | \E[0;8m |
| p8 | protect | not available |
| p9 | altcharset | ^O (off) ^N(on) |

Note: each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, since this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be '**\E[0;3;5m**'. The terminal does not have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N** depending on whether it is off or on. If all modes were to be turned on, the sequence would be '**\E[0;3;4;5;7;8m^N**'.

Now look at when different sequences are output. For example, **';3'** is output when either **'p2'** or **'p6'** is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

| sequence | when to output | terminfo translation |
|---|---|---|
| \E[0 | always | \E[0 |
| ;3 | if p2 or p6 | %?%p2%p6%\|%t;3%; |
| ;4 | if p1 or p3 or p6 | %?%p1%p3%\|%p6%\|%t;4%; |
| ;5 | if p4 | %?%p4%t;5%; |
| ;7 | if p1 or p5 | %?%p1%p5%\|%t;7%; |
| ;8 | if p7 | %?%p7%t;8%; |
| m | always | m |
| ^N or ^O | if p9 ^N, else ^O | %?%p9%t^N%e^O%; |

Putting this all together into the **sgr** sequence gives:

**sgr**=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p5%t;5%;%?%p1%p5%
|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

**Keypad**

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note: it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome** respectively. If there are function keys such as f0, f1, ..., f63, the codes they send can be given as **kf0**, **kf1**, ..., **kf63**. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0**, **lf1**, ..., **lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (BACKSPACE), **ktbc** (clear all tab stops), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. A string to program their soft-screen labels can be given as **pln**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** executes the string by the terminal in local mode; and **pfx** transmits the string to the computer. The capabilities **nlab**, **lw** and **lh** define how many soft labels there are and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln**. **smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

**Tabs and Initialization**

If the terminal has hardware tab stops, the command to advance to the next tab stop can be given as **ht** (usually CTRL-I). A "backtab" command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that TAB characters are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tab stops which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tab stops are set to. This is normally used by **'tput init'** (see **tput**(1V)) to determine whether to set the mode for hardware TAB expansion and whether to set the tab stops. If the terminal has tab stops

that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the terminal; **iprog**, the path name of a program to be run to initialize the terminal; and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **terminfo** description. They must be sent to the terminal each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tab stops using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of **tput**(1V).

Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a harder reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from **/usr/share/lib/tabset/***; however, the recommended method is to use the initialization and reset strings.) These strings are output by **'tput reset'**, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed since the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tab stops than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

If there are commands to set and clear margins, they can be given as **mgc** (clear all margins), **smgl** (set left margin), and **smgr** (set right margin).

**Delays**

Certain capabilities control padding in the terminal driver. These are primarily needed by hard-copy terminals, and are used by **'tput init'** to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

**Status Lines**

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit H19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as TAB, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, for example, **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

**Line Graphics**

If the terminal has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

| glyph name | VT100+ character |
|---|---|
| arrow pointing right | + |
| arrow pointing left | , |
| arrow pointing down | . |
| solid square block | 0 |
| lantern symbol | I |
| arrow pointing up | – |
| diamond | ‘ |
| checker board (stipple) | a |
| degree symbol | f |
| plus/minus | g |
| board of squares | h |
| lower right corner | j |
| upper right corner | k |
| upper left corner | l |
| lower left corner | m |
| plus | n |
| scan line 1 | o |
| horizontal line | q |
| scan line 9 | s |
| left tee (├) | t |
| right tee (┤) | u |
| bottom tee (┴) | v |
| top tee (┬) | w |
| vertical line | x |
| bullet | ~ |

The best way to describe a new terminal's line graphics set is to add a third column to the above table with the characters for the new terminal that produce the appropriate glyph when the terminal is in the alternate character set mode. For example,

| glyph name | VT100+ char | new tty char |
|---|---|---|
| upper left corner | l | R |
| lower left corner | m | F |
| upper right corner | k | T |
| lower right corner | j | G |
| horizontal line | q | , |
| vertical line | x | . |

Now write down the characters left to right, as in '**acsc**=lRmFkTjGq\,x.'.

**Miscellaneous**

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually CTRL-L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **'tparm(repeat_char, 'x', 10)'** is the same as **'xxxxxxxxxx'**.

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. On some UNIX systems, when the environment variable CC is set to a single-character value, all occurrences of the prototype character are replaced with that character.

Terminal descriptions that do not represent a specific kind of known terminal, such as **switch, dialup, patch,** and **network**, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to **virtual** terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt**. A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi**.

If the terminal uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not **^S** and **^Q** (CTRL-S and CTRL-Q, respectively), they may be specified with **xonc** and **xoffc**.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

**Special Cases**

The working model used by **terminfo** fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by **terminfo**. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the **terminfo** model implemented.

Terminals which can not display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals which ignore a LINEFEED immediately after an **am** wrap, such as the Concept 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive TAB characters). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie" therefore, to erase standout mode, it is instead necessary to use delete and insert

line.

Those Beehive Superbee terminals which do not transmit the escape or CTRL-C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for CTRL-C.

### Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing *xx*@ to the left of the capability definition, where *xx* is the capability. For example, the entry

**att4424-2|Teletype 4424 in display function group ii,**
**rev@, sgr@, smul@, use=att4424,**

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

## FILES

**/usr/share/lib/terminfo/?/***
compiled terminal description database

**/usr/share/lib/tabset/*** tab stop settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tab stops)

## SEE ALSO

**tput**(1V), **curses**(3V), **printf**(3S), **term**(5V), **captoinfo**(8V), **infocmp**(8V), **tic**(8V)

## WARNING

As described in the **Tabs and Initialization** section above, a terminal's initialization strings, **is1**, **is2**, and **is3**, if defined, must be output before a **curses**(3V) program is run. An available mechanism for outputting such strings is **tput init** (see **tput**(1V)).

Tampering with entries in **/usr/share/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs that expect the entry to be present and correct. In particular, removing the description for the "dumb" terminal will cause unexpected problems.

**NAME**

toc – table of contents of optional clusters in Application SunOS and Developer's Toolkit

**SYNOPSIS**

**/usr/lib/load/toc**

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

The **toc** file contains information specifying the organization of the optional clusters in Application SunOS and Developer's Toolkit on the Sun386i distribution media. For each cluster, a single line should be present with the following information:

cluster name
set containing the cluster (Application SunOS or Developer's Toolkit)
size of the cluster (in kilobytes)
diskette volume of the cluster in the set (for loading from 3.5" diskette)
tape and file number of the cluster (for loading from 1/4" tape)

Items are separated by a ':'.

Cluster names can contain any printable character other than a ':', space, tab, or newline character. The set containing the cluster is specified by an 'A' for Application SunOS or 'D' for Developer's Toolkit. The diskette volume is the number of the diskette within the diskette set on which the cluster begins. The tape and file number specifies the tape and file position of the cluster on the tape.

**EXAMPLE**

The following is an example to the **toc** file.

**accounting:A:55:14:1@12**
**advanced_admin:A:628:14:1@4**
**audit:A:144:14:1@8**
**comm:A:312:13:1@9**
**disk_quotas:A:56:14:1@11**
**doc_prep:A:790:13:1@10**
**extended_commands:A:276:13:1@5**
**games:A:2351:19:1@17**
**mail_plus:A:135:14:1@7**
**man_pages:A:5586:16:1@14**
**name_server:A:339:14:1@13**
**networking_plus:A:610:13:1@6**
**old:A:131:14:1@16**
**plot:A:227:14:1@14**
**spellcheck:A:455:13:1@2**
**sysV_commands:A:2505:14:1@3**
**base_devel:D:5389:1:2@2**
**plot_devel:D:247:5:2@3**
**sccs:D:328:5:2@4**
**sunview_devel:D:1768:5:2@5**
**sysV_devel:D:4287:3:2@6**
**proflibs:D:4755:4:2@7**
**config:D:3065:6:2@8**

The fist line specifies that the **accounting** cluster is part of Application SunOS and requires 55 kilobytes of disk storage. In the diskette distribution, it begins on diskette 14 of Application SunOS optional clusters. In the tape distribution, it can be found on file 12 of tape 1. The last line specifies that the *config* cluster is part of Developer's Toolkit and requires 3065 kilobytes of disk storage. In the diskette distribution, it begin on diskette 6 of Developer's Toolkit. In the tape distribution, it can be found on file 8 of tape 2.

**FILES**

**/usr/lib/load/toc**

**SEE ALSO**

**cluster**(1) **load**(1) **unload**(1)

**NAME**

translate – input and output files for system message translation

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

These files are used by **syslogd**(8) to translate systems messages. The input file is used to map system messages (in **printf**(3S) format strings) to numbers. This number is then used to locate a new string in the output file.

An initial part of each line in the input file may specify that the message should be suppressed. Recognized suppression specifications are:

**(NONE) Suppress the message always.**

**(n) Allow only one message every n seconds. ((10) for example).**

**() Do not suppress the message. This can be used in a message that begins with a '('.**

Note that the message suppression specification is optional. If not present, the message is not suppressed.

Each line in the output file translates the numbers from the input file into the desired error messages, and also specifies the format to be used to output each message. The order of parameters passed from the input message can be changed, by replacing the % of a format phrase with a *%num*$ where *num* is a digit string. For example, if *num* is 2, the second parameter on the input file line will be used. The value of *num* can be from 1 to the number of parameters in the input message.

If a string is translated to a number that is not found in the output file, the message is suppressed.

**EXAMPLES**

An example input file:

```
$quote "
1       "(NONE)(1) logopen test code: %s\n"
2       "(10)(2) logopen test code: %s\n"
3       "()(3) logopen test code: %s\n"
4       "()(4) logopen test code: %s\n"
5       "(10)(5) logopen testcode: %s * 100\n"
6       "(10)(6) logopen testcode: %s * 100\n"
7       "(10)(7) logopen testcode: %s * 100\n"
8       "(10)%s: %s\n"
9       "(10)\n%s: write failed, file system is full\n"
10      "(10)NFS server %s not responding still trying\n"
11      "(10)NFS %s failed for server %s: %s\n"
12      "(10)NFS server %s ok\n"
13      "(NONE)\n%s: write failed, file system is full\n"
14      "(10)NFS server %s not responding still trying\n"
15      "(100)NFS %s failed for server %s: %s\n"
```

An example output file:

```
$quote "
1       "TRANSLATION:(1) logopen test code: %s\n"
2       "TRANSLATION: (2) logopen test code: %s IS REALLY\n"
3       "TRANSLATION: (3) logopen test code: %s\n"
4       "TRANSLATION: (4) logopen test code: %s\n"
5       "TRANSLATION: (5) logopen testcode: %s * 100\n"
6       "TRANSLATION: (6) logopen testcode: %s * 100\n"
7       "TRANSLATION: (7) logopen testcode: %s * 100\n"
8       "TRANSLATION: %s: %s\n"
9       "TRANSLATION: \n%s: write failed, file system is full\n"
10      "TRANSLATION: NFS server %s not responding still trying\n"
11      "TRANSLATION: NFS %s failed for server %s: %s\n"
12      "TRANSLATION: NFS server %s ok\n"
13      "Out of disk on file system %s\n"
14      "Network file server %s not ok. Check your cable\n"
15      "Network file server %2$s down (%1$s, %3$s)\n"
```

SEE ALSO

syslogd(8)

## NAME

ttytab, ttys – terminal initialization data

## DESCRIPTION

The **/etc/ttytab** file contains information that is used by various routines to initialize and control the use of terminal special files. This information is read with the **getttyent**(3) library routines. There is one line in **/etc/ttytab** file per special file.

The **/etc/ttys** file should not be edited; it is derived from **/etc/ttytab** by **init(8)** at boot time, and is only included for backward compatibility with programs that may still require it.

Fields are separated by TAB and/or SPACE characters. Some fields may contain more than one word and should be enclosed in double quotes. Blank lines and comments can appear anywhere in the file; comments are delimited by '#' and NEWLINE. Unspecified fields default to NULL. The first field is the terminal's entry in the device directory, **/dev**. The second field of the file is the command to execute for the line, typically **getty**(8), which performs such tasks as baud-rate recognition, reading the login name, and calling **login**(1). It can be, however, any desired command, for example the start up for a window system terminal emulator or some other daemon process, and can contain multiple words if quoted. The third field is the type of terminal normally connected to that tty line, as found in the **termcap**(5) data base file. The remaining fields set flags in the **ty_status** entry (see **getttyent**(3)) or specify a window system process that **init**(8) will maintain for the terminal line.

As flag values, the strings **on** and **off** specify whether **init** should execute the command given in the second field, while **secure** in addition to **on** allows "root" to login on this line. If the console is not marked "secure," the system prompts for the root password before coming up in single-user mode. These flag fields should not be quoted. The string **window=** is followed by a quoted command string which **init** will execute before starting **getty**. If the line ends in a comment, the comment is included in the **ty_comment** field of the **ttyent** structure.

## EXAMPLE

```
console "/usr/etc/getty std.1200"  vt100       on secure
ttyd0   "/usr/etc/getty d1200"     dialup      on          # 555-1234
ttyh0   "/usr/etc/getty std.9600"  hp2621-nl   on          # 254MC
ttyh1   "/usr/etc/getty std.9600"  plugboard   on          # John's office
ttyp0   none                       network
ttyp1   none                       network     off
ttyv0   "/usr/new/xterm -L :0"     vs100       on window="/usr/new/Xvs100 0"
```

The first line permits "root" login on the console at 1200 baud, and indicates that the console is secure for single-user operation. The second example allows dialup at 1200 baud without "root" login, and the third and fourth examples allow login at 9600 baud with terminal types of **hp2621-nl** and **plugboard**, respectively. The fifth and sixth lines are examples of network pseudo-ttys, for which **getty** should not be enabled. The last line shows a terminal emulator and window-system startup entry.

## FILES

**/dev**
**/etc/ttytab**

## SEE ALSO

**login**(1), **getttyent**(3), **gettytab**(5), **termcap**(5), **getty**(8), **init**(8)

**NAME**

ttytype – data base of terminal types by port

**SYNOPSIS**

**/etc/ttytype**

**DESCRIPTION**

**ttytype** is a database containing, for each tty port on the system, the kind of terminal that is attached to it. There is one line per port, containing the terminal kind (as a name listed in **termcap**(5)), a SPACE, and the name of the tty, minus **/dev/**.

This information is read by **tset**(1) and by **login**(1) to initialize the TERM variable at login time.

**FILES**

**/dev/**

**SEE ALSO**

**login**(1), **tset**(1), **termcap**(5)

**BUGS**

Some lines are merely known as ''dialup'' or ''plugboard''.

```
extern int setjmp();
#pragma unknown_control_flow(setjmp)
#endif sparc

#endif  _TYPES_
```

The form *daddr_t* is used for disk addresses, see **fs**(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

**adb**(1), **lseek**(2), **time**(3C), **fs**(5)

NAME

tzfile – time zone information

SYNOPSIS

**#include <tzfile.h>**

DESCRIPTION

The time zone information files used by **tzset**(3V) begin with bytes reserved for future use, followed by three four-byte values of type **long**, written in a "standard" byte order (the high-order byte of the value is written first). These values are, in order:

| | |
|---|---|
| *tzh_timecnt* | The number of "transition times" for which data is stored in the file. |
| *tzh_typecnt* | The number of "local time types" for which data is stored in the file (must not be zero). |
| *tzh_charcnt* | The number of characters of "time zone abbreviation strings" stored in the file. |

The above header is followed by *tzh_timecnt* four-byte values of type **long**, sorted in ascending order. These values are written in "standard" byte order. Each is used as a transition time (as returned by **gettimeofday**(2)) at which the rules for computing local time change. Next come *tzh_timecnt* one-byte values of type **unsigned char**; each one tells which of the different types of "local time" types described in the file is associated with the same-indexed transition time. These values serve as indices into an array of *ttinfo* structures that appears next in the file; these structures are defined as follows:

```
struct ttinfo {
        long            tt_gmtoff;
        int             tt_isdst;
        unsigned int    tt_abbrind;
};
```

Each structure is written as a four-byte value for *tt_gmtoff* of type **long**, in a standard byte order, followed by a one-byte value for *tt_isdst* and a one-byte value for *tt_abbrind*. In each structure, *tt_gmtoff* gives the number of seconds to be added to GMT, *tt_isdst* tells whether *tm_isdst* should be set by **localtime** (see **ctime**(3)) and *tt_abbrind* serves as an index into the array of time zone abbreviation characters that follow the *ttinfo* structure(s) in the file.

**localtime** uses the first standard-time *ttinfo* structure in the file (or simply the first *ttinfo* structure in the absence of a standard-time structure) if either *tzh_timecnt* is zero or the time argument is less than the first transition time recorded in the file.

SEE ALSO

**gettimeofday**(2), **ctime**(3), **localtime**(3), **tzset**(3V)

**ll_time** **long** containing the time at which the user logged in, in seconds since 00:00 GMT, January 1, 1970.

**ll_line** Character array containing the name of the terminal on which the user logged in.

**ll_host** Character array containing the name of the host from which the user remotely logged in, if they logged in from another host; otherwise, a null string.

When reporting (and updating) the most recent login date, **login** performs an **lseek**(2) to a byte-offset in **/var/adm/lastlog** corresponding to the userid. Because the count of userids may be high, whereas the number actual users may be small within a network environment, the bulk of this file may never be allocated by the file system even though an offset may appear to be quite large. Although **ls**(1V) may show it to be large, chances are that this file need not truncated. **du**(1V) will report the correct (smaller) amount of space actually allocated to it.

**FILES**

**/etc/utmp**
**/var/adm/wtmp**
**/var/adm/lastlog**

**SEE ALSO**

**login**(1), **who**(1), **ac**(8), **init**(8)

## NAME

uuencode – format of an encoded uuencode file

## DESCRIPTION

Files output by **uuencode**(1C) consist of a header line, followed by a number of body lines, and a trailer line. **uudecode** (see **uuencode**(1C)) will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters '**begin** '. The word **begin** is followed by a mode (in octal), and a string which names the remote file. Spaces separate the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing NEWLINE). These consist of a character count, followed by encoded characters, followed by a NEWLINE. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a SPACE to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII SPACE.

The trailer line consists of **end** on a line by itself.

## SEE ALSO

**mail**(1), **uuencode**(1C), **uucp**(1C), **uusend**(1C)

## NAME

vfont – font formats

## SYNOPSIS

**#include <vfont.h>**

## DESCRIPTION

The fonts used by the window system and printer/plotters have the following format. Each font is in a file, which contains a header, an array of character description structures, and an array of bytes containing the bit maps for the characters. The header has the following format:

```
struct header {
        short           magic;          /* Magic number VFONT_MAGIC */
        unsigned short  size;           /* Total # bytes of bitmaps */
        short           maxx;           /* Maximum horizontal glyph size */
        short           maxy;           /* Maximum vertical glyph size */
        short           xtend;          /* (unused) */
};
#define VFONT_MAGIC                     0436
```

*maxx* and *maxy* are intended to be the maximum horizontal and vertical size of any glyph in the font, in raster lines. (A glyph is just a printed representation of a character, in a particular size and font.) The **size** is the total size of the bit maps for the characters in bytes. The *xtend* field is not currently used.

After the header is an array of NUM_DISPATCH structures, one for each of the possible characters in the font. Each element of the array has the form:

```
struct dispatch {
        unsigned short  addr;           /* &(glyph) - &(start of bitmaps) */
        short           nbytes;         /* # bytes of glyphs (0 if no glyph) */
        char            up, down, left, right;  /* Widths from baseline point */
        short           width;          /* Logical width, used by troff */
};
#define NUM_DISPATCH                    256
```

The *nbytes* field is nonzero for characters which actually exist. For such characters, the *addr* field is an offset into the bit maps to where the character's bit map begins. The *up* , *down*, *left*, and *right* fields are offsets from the base point of the glyph to the edges of the rectangle which the bit map represents. (The imaginary "base point" is a point which is vertically on the "base line" of the glyph (the bottom line of a glyph which does not have a descender) and horizontally near the left edge of the glyph; often 3 or so pixels past the left edge.) The bit map contains *up+down* rows of data for the character, each of which has *left+right* columns (bits). Each row is rounded up to a number of bytes. The *width* field represents the logical width of the glyph in bits, and shows the horizontal displacement to the base point of the next glyph.

## FILES

**/usr/lib/vfont/***
**/usr/lib/fonts/fixedwidthfonts/***

## SEE ALSO

**troff**(1), **vfontinfo**(1), **vswap**(1)

## BUGS

A machine-independent font format should be defined. The **shorts** in the above structures contain different bit patterns depending whether the font file is for use on a VAX or a Sun. The **vswap** program must be used to convert one to the other.

## NAME

vgrindefs – vgrind's language definition data base

## SYNOPSIS

**/usr/lib/vgrindefs**

## DESCRIPTION

**vgrindefs** contains all language definitions for vgrind. The data base is very similar to **termcap(5)**. Capabilities in **vgrindefs** are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list. Entries may continue onto multiple lines by giving a \ as the last character of a line. Lines starting with # are comments.

### Capabilities

The following table names and describes each capability.

| Name | Type | Description |
|---|---|---|
| **ab** | str | Regular expression for the start of an alternate form comment |
| **ae** | str | Regular expression for the end of an alternate form comment |
| **bb** | str | Regular expression for the start of a block |
| **be** | str | Regular expression for the end of a lexical block |
| **cb** | str | Regular expression for the start of a comment |
| **ce** | str | Regular expression for the end of a comment |
| **id** | str | String giving characters other than letters and digits that may legally occur in identifiers (default '_') |
| **kw** | str | A list of keywords separated by spaces |
| **lb** | str | Regular expression for the start of a character constant |
| **le** | str | Regular expression for the end of a character constant |
| **oc** | bool | Present means upper and lower case are equivalent |
| **pb** | str | Regular expression for start of a procedure |
| **pl** | bool | Procedure definitions are constrained to the lexical level matched by the 'px' capability |
| **px** | str | A match for this regular expression indicates that procedure definitions may occur at the next lexical level. Useful for lisp-like languages in which procedure definitions occur as subexpressions of defuns. |
| **sb** | str | Regular expression for the start of a string |
| **se** | str | Regular expression for the end of a string |
| **tc** | str | Use the named entry as a continuation of this one |
| **tl** | bool | Present means procedures are only defined at the top lexical level |

### Regular Expressions

**vgrindefs** uses regular expressions similar to those of **ex(1)** and **lex(1)**. The characters '^', '$', ':', and '\' are reserved characters and must be 'quoted' with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

| | |
|---|---|
| $ | The end of a line |
| ^ | The beginning of a line |
| \d | A delimiter (space, tab, newline, start of line) |
| \a | Matches any string of symbols (like '.*' in lex) |
| \p | Matches any identifier. In a procedure definition (the 'pb' capability) the string that matches this symbol is used as the procedure name. |
| () | Grouping |
| \| | Alternation |
| ? | Last item is optional |
| \e | Preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) that can include the string delimiter in a string by escaping it. |

Unlike other regular expressions in the system, these match words and not characters. Hence something like '(tramp|steamer)flies?' would match 'tramp', 'steamer', 'trampflies', or 'steamerflies'. Contrary to some forms of regular expressions, vgrindef alternation binds very tightly. Grouping parentheses are likely to be necessary in expressions involving alternation.

**Keyword List**

The keyword list is just a list of keywords in the language separated by spaces. If the 'oc' boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

## EXAMPLE

The following entry, which describes the C language, is typical of a language entry.

```
C|c|the C programming language:\
        :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
        :lb=':le=\e':tl:\
        :kw=asm auto break case char continue default do double else enum\
        extern float for fortran goto if int long register return short\
        sizeof static struct switch typedef union unsigned while #define\
        #else #endif #if #ifdef #ifndef #include #undef # define else endif\
        if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to **vgrind**(1) as 'c' or 'C'.

## FILES

**/usr/lib/vgrindefs** file containing terminal descriptions

## SEE ALSO

**vgrind**(1), **troff**(1)

## NAME

ypfiles – the Yellow Pages database and directory structure

## DESCRIPTION

The Yellow Pages (YP) network lookup service uses a distributed, replicated database of **dbm** files contained in the **/var/yp** directory hierarchy on each YP server. A **dbm** database consists of two files, created by calls to the **ndbm**(3) library package. One has the filename extension **.pag** and the other has the filename extension **.dir**. For instance, the database named **hosts.byname**, is implemented by the pair of files **hosts.byname.pag** and **hosts.byname.dir**.

A **dbm** database served by the YP is called a YP *map*. A YP *domain* is a subdirectory of **/var/yp** containing a set of YP maps. Any number of YP domains can exist. Each may contain any number of maps.

No maps are required by the YP lookup service itself, although they may be required for the normal operation of other parts of the system. There is no list of maps which YP serves — if the map exists in a given domain, and a client asks about it, the YP will serve it. For a map to be accessible consistently, it must exist on all YP servers that serve the domain. To provide data consistency between the replicated maps, an entry to run **ypxfr** periodically should be made in the super-user's **crontab** file on each server. More information on this topic is in **ypxfr**(8).

YP maps should contain two distinguished key-value pairs. The first is the key **YP_LAST_MODIFIED**, having as a value a ten-character ASCII order number. The order number should be the system time in seconds when the map was built. The second key is **YP_MASTER_NAME**, with the name of the YP master server as a value. **makedbm**(8) generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the YP, but the **ypserv** process will not be able to return values for "Get order number" or "Get master name" requests. See **ypserv**(8). In addition, values of these two keys are used by **ypxfr** when it transfers a map from a master YP server to a slave. If **ypxfr** cannot figure out where to get the map, or if it is unable to determine whether the local copy is more recent than the copy at the master, you must set extra command line switches when you run it.

YP maps must be generated and modified only at the master server. They are copied to the slaves using **ypxfr**(8) to avoid potential byte-ordering problems among YP servers running on machines with different architectures, and to minimize the amount of disk space required for the dbm files. The YP database can be initially set up for both masters and slaves by using **ypinit**(8).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the YP map and is propagated from the master to the slaves by running **/var/yp/Makefile**. All Sun-supplied maps have entries in **/var/yp/Makefile**; if you add a YP map, edit this file to support the new map. The makefile uses **makedbm**(8) to generate the YP map on the master, and **yppush**(8) to propagate the changed map to the slaves. **yppush** is a client of the map **ypservers**, which lists all the YP servers. For more information on this topic, see **yppush**(8).

## FILES

**/var/yp**
**/var/yp/Makefile**

## SEE ALSO

**dbm**(3X), **makedbm**(8), **rpcinfo**(8C), **ypinit**(8), **ypmake**(8), **yppoll**(8), **yppush**(8), **ypserv**(8), **ypxfr**(8),

**NAME**

intro – introduction to games and demos

**DESCRIPTION**

This section describes available games and demos.

**LIST OF GAMES AND DEMOS**

| Name | Appears on Page | Description |
|---|---|---|
| **adventure** | **adventure**(6) | an exploration game |
| **arithmetic** | **arithmetic**(6) | provide drill in number facts |
| **backgammon** | **backgammon**(6) | the game of backgammon |
| **banner** | **banner**(6) | print large banner on printer |
| **battlestar** | **battlestar**(6) | a tropical adventure game |
| **bcd** | **bcd**(6) | convert to antique media |
| **bj** | **bj**(6) | the game of black jack |
| **boggle** | **boggle**(6) | play the game of boggle |
| **boggletool** | **boggletool**(6) | play a game of boggle |
| **bouncedemo** | **graphics_demos**(6) | graphics demonstration programs |
| **canfield** | **canfield**(6) | Canfield solitaire card game |
| **canfieldtool** | **canfield**(6) | Canfield solitaire card game |
| **canvas_demo** | **sunview_demos**(6) | Window-System demonstration programs |
| **cfscores** | **canfield**(6) | Canfield solitaire card game |
| **chase** | **chase**(6) | try to escape to killer robots |
| **chess** | **chess**(6) | the game of chess |
| **chesstool** | **chesstool**(6) | window-based front-end to chess program |
| **ching** | **ching**(6) | the book of changes and other cookies |
| **craps** | **craps**(6) | the game of craps |
| **cribbage** | **cribbage**(6) | the card game cribbage |
| **cursor_demo** | **sunview_demos**(6) | Window-System demonstration programs |
| **factor** | **factor**(6) | factor a number, generate large primes |
| **fish** | **fish**(6) | play "Go Fish" |
| **fortune** | **fortune**(6) | print a random, hopefully interesting, adage |
| **framedemo** | **graphics_demos**(6) | graphics demonstration programs |
| **gammontool** | **gammontool**(6) | play a game of backgammon |
| **graphics_demos** | **graphics_demos**(6) | graphics demonstration programs |
| **hack** | **hack**(6) | replacement for rogue |
| **hangman** | **hangman**(6) | computer version of the game hangman |
| **hunt** | **hunt**(6) | a multiplayer multiterminal game |
| **jumpdemo** | **graphics_demos**(6) | graphics demonstration programs |
| **life** | **life**(6) | John Conway's game of life |
| **mille** | **mille**(6) | play Mille Bornes |
| **monop** | **monop**(6) | Monopoly game |
| **moo** | **moo**(6) | guessing game |
| **number** | **number**(6) | convert Arabic numerals to English |
| **ppt** | **bcd**(6) | convert to antique media |
| **primes** | **factor**(6) | factor a number, generate large primes |
| **primes** | **primes**(6) | print all primes larger than some given number |
| **quiz** | **quiz**(6) | test your knowledge |
| **rain** | **rain**(6) | animated raindrops display |
| **random** | **random**(6) | select lines randomly from a file |
| **robots** | **robots**(6) | fight off villainous robots |
| **snake** | **snake**(6) | display chase game |
| **snscore** | **snake**(6) | display chase game |
| **spheresdemo** | **graphics_demos**(6) | graphics demonstration programs |

| | | |
|---|---|---|
| **sunview_demos** | **sunview_demos**(6) | Window-System demonstration programs |
| **trek** | **trek**(6) | trekkie game |
| **worm** | **worm**(6) | play the growing worm game |
| **worms** | **worms**(6) | animate worms on a display terminal |
| **wump** | **wump**(6) | the game of hunt-the-wumpus |

**NAME**

adventure – an exploration game

**SYNOPSIS**

**/usr/games/adventure**

**DESCRIPTION**

The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type **quit**; to save a game for later resumption, type **suspend**.

**BUGS**

Saving a game creates a large executable file instead of just the information needed to resume the game.

**NAME**

arithmetic – provide drill in number facts

**SYNOPSIS**

**/usr/games/arithmetic** [ +–x/ ] [ *range* ]

**DESCRIPTION**

**arithmetic** types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (DELETE character).

The first optional argument determines the kind of problem to be generated; '+', '–', 'x', '/' respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +–.

*range* is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

## NAME

backgammon – the game of backgammon

## SYNOPSIS

**backgammon** [ – ] [ n r w b pr pw pb t**term** s*filename* ]

## DESCRIPTION

**backgammon** lets you play backgammon against the computer or against a 'friend'. All commands only are one letter, so you don't need to type a carriage return, except at the end of a move. **backgammon** is mostly self documenting, so that a q ? (question mark) will usually get some help. If you answer **y** when **backgammon** asks if you want the rules, you will get text explaining the rules of the game, some hints on strategy, instruction on how to use **backgammon,** and a tutorial consisting of a practice game against the computer. A description of how to use **backgammon** can be obtained by answering **y** when it asks if you want instructions. The possible arguments for **backgammon** (most are unnecesary but some are very convenient) consist of:

- **n** don't ask for rules or instructions
- **r** player is red (implies n)
- **w** player is white (implies n)
- **b** two players, red and white (implies n)
- **pr** print the board before red's turn
- **pw** print the board before white's turn
- **pb** print the board before both player's turn
- t*term* terminal is type *term*, uses */etc/termcap*, otherwise uses the TERM environment variable.
- s*file* recover previously saved game from *file*. This can also be done by executing the saved file, that is, typing its name in as a command.

Arguments may be optionally preceded by a – sign. Several arguments may be concatenated together, but not after **s** or **t** arguments, since they can be followed by an arbitrary string. Any unrecognized arguments are ignored. An argument of a lone – gets a description of possible arguments.

If **term** has capabilities for direct cursor movement. **backgammon** 'fixes' the board after each move, so the board does not need to be reprinted, unless the screen suffers some horrendous malady. Also, any 'p' option will be ignored.

## QUICK REFERENCE

When **backgammon** prompts by typing only your color, type a space or carriage return to roll, or

- **d** to double
- **p** to print the board
- **q** to quit
- **s** to save the game for later

When **backgammon** prompts with 'Move:', type

- **p** to print the board
- **q** to quit
- **s** to save the game

or a *move*, which is a sequence of

- **s-f** move from **s** to **f**
- **s/r** move one man on **s** the roll **r** separated by commas or spaces and ending with a newline. Available abbreviations are

**s-f1-f2** means **s-f1,f1-f2**

**s/r1r2** means **s/r1,s/r2**

Use **b** for bar and **h** for home, or **0** or **25** as appropriate.

FILES

| | |
|---|---|
| **/usr/games/teachgammon** | rules and tutorial |
| **/etc/termcap** | terminal capabilities |

BUGS

**backgammon**'s strategy needs much work.

## NAME

banner – print large banner on printer

## SYNOPSIS

**/usr/games/banner** [ –w*n* ] message ...

## DESCRIPTION

**banner** prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If –**w** is given, the output is reduced from a width of 132 to *n*, suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

## BUGS

Several ASCII characters are not defined, notably '<', '>', '[', ']', '\', '^', '_', '{', '}', '|', and '~'.
Also, the characters '"', ''', and '&' are funny looking (but in a useful way.)

The –**w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

## NAME

battlestar – a tropical adventure game

## SYNOPSIS

**battlestar** [ –**r** ]

## DESCRIPTION

**battlestar** is an adventure game in the classic style. However, it is slightly less of a puzzle and more a game of exploration. There are a few magical words in the game, but on the whole, simple English should suffice to make one's desires understandable to the parser.

## OPTIONS

–**r** Recover a saved game.

## THE SETTING

In the days before the darkness came, when battlestars ruled the heavens...

Three He made and gave them to His daughters,
Beautiful nymphs, the goddesses of the waters.
One to bring good luck and simple feats of wonder,
Two to wash the lands and churn the waves asunder,
Three to rule the world and purge the skies with thunder.

In those times great wizards were known and their powers were beyond belief. They could take any object from thin air, and, uttering the word '**su**', could disappear.

In those times men were known for their lust of gold and desire to wear fine weapons. Swords and coats of mail were fashioned that could withstand a laser blast.

But when the darkness fell, the rightful reigns were toppled. Swords and helms and heads of state went rolling across the grass. The entire fleet of battlestars was reduced to a single ship.

## USAGE

### Sample Commands

| | | |
|---|---|---|
| take | --- | take an object |
| drop | --- | drop an object |
| wear | --- | wear an object you are holding |
| draw | --- | carry an object you are wearing |
| puton | --- | take an object and wear it |
| take off | --- | draw an object and drop it |

throw <object> <direction>
! <shell esc>

### Implied Objects

```
>-: take watermelon
watermelon:
Taken.
>-: eat
watermelon:
Eaten.
>-: take knife and sword and apple, drop all
knife:
Taken.
broadsword:
Taken.
apple:
Taken.
knife:
Dropped.
```

broadsword:
Dropped.
apple:
Dropped.
>-: get
knife:
Taken.

Notice that the "shadow" of the next word stays around if you want to take advantage of it. That is, saying '**take knife**' and then '**drop**' will drop the knife you just took.

**Score and Inven**

The two commands **score** and **inven** will print out your current status in the game.

**Saving a Game**

The command **save** will save your game in a file called **Bstar**. You can recover a saved game by using the **–r** option when you start up the game.

**Directions**

The compass directions N, S, E, and W can be used if you have a compass. If you do not have a compass, you will have to say **R**, **L**, **A**, or **B**, which stand for Right, Left, Ahead, and Back. Directions printed in room descriptions are always printed in R, L, A, & B relative directions.

**BUGS**

Countless.

**NAME**

bcd, ppt – convert to antique media

**SYNOPSIS**

**/usr/games/bcd** *text*

**/usr/games/ppt**

**DESCRIPTION**

**bcd** converts the literal *text* into a form familiar to old-timers.

**ppt** converts the standard input into yet another form.

**SEE ALSO**

**dd**(1)

**NAME**

bj – the game of black jack

**SYNOPSIS**

**/usr/games/bj**

**DESCRIPTION**

**bj** is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

The bet is $2 every hand.

A player "natural" (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a "push" (no money exchange).

If the dealer has an ace up, the player is allowed to make an "insurance" bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.

If the player is dealt two cards of the same value, he is allowed to "double". He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)

If a dealt hand has a total of ten or eleven, the player may "double down". He may double the bet ($2 to $4) and receive exactly one more card on that hand.

Under normal play, the player may "hit" (draw a card) as long as his total is not over twenty-one. If the player "busts" (goes over twenty-one), the dealer wins the bet.

When the player "stands" (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.

If both player and dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions will be asked at appropriate times. Each question is answered by y followed by a new-line for "yes", or just new-line for "no".

**?** (this means, "do you want a hit?")
**Insurance?**
**Double down?**

Every time the deck is shuffled, the dealer so states and the "action" (total bet) and "standing" (total won or lost) is printed. To exit, hit the interrupt key (CTRL–C) and the action and standing will be printed.

**NAME**

boggle – play the game of boggle

**SYNOPSIS**

**/usr/games/boggle** [ + ] [ ++ ]

**AVAILABILITY**

This game is available with the *Games* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words.**

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

## NAME

boggletool – play a game of boggle

## SYNOPSIS

**/usr/games/chesstool** [ *number* ] [ +[+]] [ 16-character *string* ]

## AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**boggletool** allows you to play the game of Boggle (TM Parker Bros.) against the computer. The *number* argument specifies the time limit in minutes (the default is 3 minutes). If a 16 character long string is placed on the command line, it is interpreted as a Boggle board: the first four letters form the top row, the next four letters the second row, etc. If no letters are specified, a board is randomly rolled by the computer from a set of Boggle cubes. The +[+] argument is explained below under **Advanced Play .**

## PLAYING THE GAME

### Rules of the Game

The object of Boggle is to find as many words as possible in a 4 by 4 grid of letters within a certain time limit. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. Normally, no letter in the grid may be used more than once in a word (see **Advanced Play** for exceptions).

### Playing the Game

When invoked, boggletool displays a grid of letters and an hourglass. To enter words, simply type in lower case letters to spell the word you want. Use any whitespace (SPACE, TAB, or NEWLINE) to finish a word. To correct mistakes you make, use BACKSPACE or DEL to delete the last character, or use CTRL-U to delete an entire word. **boggletool** verifies that words you enter are both in the grid and are valid English words. If you type in a character which would form a word which is not in the grid, the display will flash and the character you typed will not be echoed. When you type any whitespace to end the current word, **boggletool** will verify that the word is three or more letters long and that it appears in the dictionary. If the word you typed is illegal for either reason, the display will flash and you will have to either erase the word or change it. If you try to enter a valid word which you have already entered, the display will flash and the previous occurrence of the word will be highlighted. Again, you will have to erase the word before continuing. As you enter words, the "sand" in the hourglass will fall. At the end of the time limit, the display will flash and you will no longer be allowed to enter words. After a moment, the computer will display two lists of words: the words you found, and other words which also appear in the grid. To play another game, just type any capital letter (or use the pop-up menu).

### Using the Menu

The pop-up menu is invoked by pressing the RIGHT mouse button. There are four items in it, and they work as follows.

**Restart Game**
: Create a new boggletool a new board, reset the timer, and allow you to start from scratch.

**Restart Timer**
: Allows you to cheat by reseting the hourglass timer to zero.

**Give Up**
: End the game and print the results immediately.

**Quit** Allows you to quit running the boggletool program. A prompt appears asking you to confirm the quit; when it does, click the LEFT mouse button to quit or the RIGHT mouse button to abort the quit.

### Advanced Play

There are two options for advanced players. If a single + appears on the command line, letters in the grid may be reused. If two +'s are on the command line, letters may also be considered adjacent to themselves

as well as to their neighbors. Although it is far easier to find words with these two options, there are also many more possible words in the grid and it is therefore difficult to find them all.

**FILES**

**/usr/games/boggledict** dictionary file for computer's words

## NAME

canfield, canfieldtool, cfscores – Canfield solitaire card game

## SYNOPSIS

**/usr/games/canfield** [ **–ac** ]

**/usr/games/canfieldtool** [ **–ac** ]

**/usr/games/cfscores** [ **–ac** ] [ *username* ]

## AVAILABILITY

These games are available with the *Games* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**canfield** can be played on any terminal. **canfieldtool** is the SunView version with attractive graphics.

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In **canfield**, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types **ht** for his move. Foundation base cards are also automatically moved to the foundation when they become available.

### Canfieldtool

Once you understand the rules, **canfieldtool** is self-explanatory.

### Canfield

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (The initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

## OPTIONS

**a** Print out **canfield** accounts for all users that have played the game since the database was set up.

**c** Maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase the chances of winning.

With no arguments, **cfscores** prints out the current status of your canfield account. If *username* is specified, it prints out the status of their account.

## FILES

**/usr/games/canfield** the game itself
**/usr/games/lib/cfscores** the database of scores

## BUGS

It is impossible to cheat.

**NAME**

chase – try to escape to killer robots

**SYNOPSIS**

**/usr/games/chase** [ *nrobots* ] [ *nfences* ]

**DESCRIPTION**

The object of the game **chase** is to move around inside of the box on the screen without getting eaten by the robots chasing and without running into anything.

If a robot runs into another robot while chasing you, they crash and leave a junk heap. If a robot runs into a fence, it is destroyed.

If you can survive until all the robots are destroyed, you have won!

If you do not specify either *nrobots* or *nfences*, chase will prompt you for them.

## NAME

chess – the game of chess

## SYNOPSIS

**/usr/games/chess**

## AVAILABILITY

This game is available for Sun-2, Sun-3 and Sun-4 systems with the *Games* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**chess** is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '**+**' is used to specify check; '**o-o**' and '**o-o-o**' specify castling. To play black, type '**first**'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

## DIAGNOSTICS

The most cryptic diagnostic is '**eh?**' which means that the input was syntactically incorrect.

## FILES

**/usr/games/lib/chess.book**
book of opening moves

## BUGS

Pawns may be promoted only to queens.

**NAME**

chesstool – window-based front-end to chess program

**SYNOPSIS**

**/usr/games/chesstool** [ *chess_program* ]

**AVAILABILITY**

This game is available for Sun-2, Sun-3 and Sun-4 systems, with the *Games* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**chesstool** is a window-based front-end to the **chess**(6) program. Used without options, **chesstool** uses **/usr/games/chess**; you can designate any alternate program which uses the same command syntax as **chess**(6) with the *chess_program* argument.

When **chesstool** starts up, it displays a large window with three subwindows. The first subwindow displays messages '**Illegal move**', for example. The second subwindow is an options subwindow; options are described below. The final subwindow is a chessboard display with white and black pieces and two (advisory only) timekeeping clocks.

Make your moves with the mouse: select a piece by positioning the arrow cursor over the piece and pressing the left mouse button down, then drag the piece to the destination square, and release the button. The cursor will then turn to an hourglass icon while the system plays.

Items in the subwindow may be selected with either the left or middle mouse buttons. These options are:

**Last Play** Show the last play made.

**Undo** Undo your last move and the machine's response.

Once the game is over, it is not possible to restart it, so undo will update the board, but the game cannot be continued from that position.

**Flash** Flash when the machine has completed its move.

When this command is selected, a check mark will appear next to the word **Flash**. In flash mode, if **chesstool** is open, the piece moved by the system on its play will flash until you make your move. If **chesstool** is iconic, the entire icon will flash when the machine has made its move. Thus you can "Close" **chesstool** and be alerted when it's your turn to move. To turn flash mode off, select flash again.

**Machine White** Start a new game with the machine playing white.

**Human White** Start a new game with the machine playing black.

**Quit** Exit from **chesstool**.

There are two moves which are special: castling and capturing a pawn *en*passant. To castle, move the king only. The position of the rook will automatically be updated. Since the king moves two squares when castling, the move is unambiguous. To capture *en*passant, move the pawn to the square occupied by the opposing pawn which will be captured.

**SEE ALSO**

**chess**(6)

**NAME**

ching – the book of changes and other cookies

**SYNOPSIS**

**/usr/games/ching** [*hexagram*]

**DESCRIPTION**

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams*, each symbolized by a particular arrangement of six straight (——) and broken (– –) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each *hexagram* consists of two major sections. The **Judgement** relates specifically to the matter at hand (For instance, "It furthers one to have somewhere to go.") while the **Image** describes the general attributes of the *hexagram* and how they apply to one's own life ("Thus the superior man makes himself strong and untiring.").

When any of the lines has the value six or nine, it is a moving line; for any such line there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second *hexagram* (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow–stalks or tossed coins. The resulting *hexagram* will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, this oracle simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process ID and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin–toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try **fortune**(6).

**SEE ALSO**

It furthers one to see the great man.

**DIAGNOSTICS**

The great prince issues commands,
Founds states, vests families with fiefs.
Inferior people should not be employed.

**BUGS**

Waiting in the mud
Brings about the arrival of the enemy.

If one is not extremely careful,
Somebody may come up from behind and strike him.
Misfortune.

NAME

craps – the game of craps

SYNOPSIS

**/usr/games/craps**

DESCRIPTION

**craps** is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while the user (the *player*) places bets. The player may choose, at any time, to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a "bankroll" of $2,000.

The program prompts with:

**bet?**

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with **bet?** until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The *first* roll is the roll immediately following a bet:

1. On the first roll:

| | |
|---|---|
| 7 or 11 | wins for the roller; |
| 2, 3, or 12 | wins for the House; |
| any other number | is the *point*, roll again (Rule 2 applies). |

2. On subsequent rolls:

| | |
|---|---|
| point | roller wins; |
| 7 | House wins; |
| any other number | roll again. |

If a player loses the entire bankroll, the House will offer to lend the player an additional $2,000. The program will prompt:

**marker?**

A **yes** (or **y**) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

**Repay marker?**

A reply of **yes** (or **y**) indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

**How many?**

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program will prompt with **How many?** until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the *total* amount of money borrowed will be *automatically* repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

**New game?**

to give the House a chance to win back its money.

Any reply other than **yes** is considered to be a **no** (except in the case of **bet?** or **How many?**). To exit, send an interrupt (break), DELETE character or CTRL–D The program will indicate whether the player won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

## NAME

cribbage – the card game cribbage

## SYNOPSIS

**/usr/games/cribbage** [ **–eqr** ] *name* ...

## DESCRIPTION

**cribbage** plays the card game cribbage, with **cribbage** playing one hand and the user the other. **cribbage** initially asks the user if the rules of the game are needed – if so, **cribbage** displays the appropriate section from *According to Hoyle* with **more**(1).

## OPTIONS

**–e** Provide an explanation of the correct score when the player makes mistakes scoring his hand or crib. This is especially useful for beginning players.

**–q** Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

**–r** Instead of asking the player to cut the deck, **cribbage** will randomly cut the deck.

## PLAYING CRIBBAGE

**cribbage** first asks the player whether he wishes to play a short game ("once around", to 61) or a long game ("twice around", to 121). A response of 's' results in a short game, any other response plays a long game.

At the start of the first game, **cribbage** asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, **cribbage** first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, **cribbage** cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. **cribbage** keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. **cribbage** requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

## SPECIFYING CARDS

Cards are specified as *rank* followed by *suit*. The *rank*s may be specified as one of **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **t**, **j**, **q**, and **k**, or alternatively, one of **ace**, **two**, **three**, **four**, **five**, **six**, **seven**, **eight**, **nine**, **ten**, **jack**, **queen**, and **king**. *Suit*s may be specified as **s**, **h**, **d**, and **c**, or alternatively as **spades**, **hearts**, **diamonds**, and **clubs**. A card may be specified as *rank suit*, or *rank* of *suit*. If the single letter *rank* and *suit* designations are used, the space separating the *suit* and *rank* may be left out. Also, if only one card of the desired *rank* is playable, typing the *rank* is sufficient. For example, if your hand was **2h**, **4d**, **5c**, **6h**, **jc**, **kd** and you wanted to discard the king of diamonds, you could type any of **k**, **king**, **kd**, **k d**, **k of d**, **king d**, **king of d**, **k diamonds**, **k of diamonds**, **king diamonds**, or **king of diamonds**,

## FILES

**/usr/games/cribbage**

**SEE ALSO**

**more**(1)

## NAME

factor, primes – factor a number, generate large primes

## SYNOPSIS

**/usr/games/factor** [ *number* ]

**/usr/games/primes** [ *number* ]

## DESCRIPTION

**factor** reads lines from its standard input. If it reads a positive number, **factor** will factor the number and print its prime factors, printing each one the proper number of times. **factor** exits when it reads zero, a negative number, or something other than a number. If a *number* is given, **factor** will factor the number, print its prime factors, and exit.

**primes** reads a number from the standard input and prints all primes larger than the given number and smaller than $2^{32}$ (about $4.3 \times 10^9$). If a *number* is given, **primes** will use that number rather than reading one from the standard input.

## DIAGNOSTICS

**Ouch.** Input out of range or for garbage input.

**NAME**

fish – play "Go Fish"

**SYNOPSIS**

**/usr/games/fish**

**DESCRIPTION**

**fish** plays the game of "Go Fish", a children's card game. The object is to accumulate "books" of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the "pool" of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **j**, **q**, or **k** k when asked. Hitting a RETURN character gives you information about the size of my hand and the pool, and tells you about my books. Saying '**p**' as a first guess puts you into "pro" level; the default is pretty dumb.

**NAME**

fortune – print a random, hopefully interesting, adage

**SYNOPSIS**

**/usr/games/fortune** [ – ] [ –**alsw** ] [ *filename* ]

**DESCRIPTION**

**fortune** with no arguments prints out a random adage. The flags mean:

–**a** Choose from either list of adages.

–**l** Long messages only.

–**s** Short messages only.

–**w** Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

**FILES**

**/usr/games/lib/fortunes.dat**

## NAME

gammontool – play a game of backgammon

## SYNOPSIS

**/usr/games/gammontool** [ *path* ]

## AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**gammontool** paints a backgammon board on the screen, and then lets you play against the computer. It must be run in SunWindows. The optional *path* argument specifies an alternate move-generating program, which must be specially designed to run with **gammontool.**

The game has three subwindows: an option window on top, a message window in the middle, and a large board on the bottom. The buttons in the option window are used to restart, double, etc. The message window has two lines: the first tells whose turn it is, and the second displays any errors that occur.

### The Initial Roll

To start the game, roll the dice to determine who goes first. Move the mouse arrow onto the board and click the left button. One die appears on each side of the board: the die on the left is yours, and the die on the right is the computer's. If your roll is greater, then you move; if not, the computer makes a move.

### Making Your Move

When it is your turn, '**Your**move' appears in the message window. Place the mouse over any piece of your color, and click the left button. While holding down the button, move the mouse to drag the piece; the piece follows the mouse until you release the button. The tool checks each move and does not allow illegal moves. When you have made as many moves as you can, the computer takes its turn; after it finishes, you may either roll again, or double.

*Doubling*

To double, click the *Double* button in the option window and wait for the computer's response. If the computer doubles you, a message is displayed and you must answer with the **Accept Double** or **Refuse Double** buttons. The **Forfeit** button can also be used to refuse a double. If the game is doubled, a doubling cube with the proper value is displayed on the bar strip. If the number is facing up, then you may double next. If the number is upside down, it is the computer's turn to double.

*Other Buttons*

If you want to change your move before you have finished it, use the **Redo Move** or **Redo Entire Move** buttons in the option window. **Redo Entire Move** replaces all of the pieces you have moved so that you can redo them all. **Redo Move** only replaces the last piece you moved, so it is useful when you roll doubles and want to redo only the last piece you moved. Note that once you have made all of the moves your roll permits, play passes immediately to the computer, so you cannot redo the very last move. The **Show Last Move** button allows you to see the last move again.

### Leaving the Game

If you want to quit playing backgammon, use the **Quit** button. If you want to forfeit the game, use the **Forfeit** button. The computer penalizes you by taking a certain number of points, but the program does not terminate.

To play another game after winning, losing, or forfeiting, click the **New Game** button. To change the color of your pieces, click the mouse button while pointing at either the **White** or **Black** checkboxes. You may change colors at any time, even in the middle of a game. Changing colors in the middle of a game does not mean that you trade places with the computer; your pieces stay where they are, but they are repainted with the new color. Your pieces always move from the top right to the bottom right of the board, regardless of your color. As an additional cue as to your color, your dice are always displayed on the left half of the board.

**Log File**

If a there is a **gammonlog** file your home directory, **gammontool** keeps a log of the games played. Each move and double gets recorded, along with the winners and accumulated scores.

## FILES

| | |
|---|---|
| **~/gammonlog** | log of games played |
| **/usr/games/lib/gammonscores** | log of wins and losses |

## BUGS

The default strategy used by the computer is very poor.

If a single move uses more than one die (for instance if you roll 5, 6 and move 11 spaces without touching down in the middle) it is unpredictable where the program will make the piece touch down. This may be important if there is a blot on one of these middle points. The program will always make the move if possible, but if two midpoints would work and there is a blot on one of them, it is much better to explicitly hit the blot and then move the piece the rest of the way.

## NAME

graphics_demos, bouncedemo, framedemo, jumpdemo, spheresdemo, – graphics demonstration programs

## SYNOPSIS

**/usr/demo/bouncedemo** [ **–d** *dev* ] [ **–n***x* ] [ **–r** ] [ **–q** ] **/usr/demo/framedemo** [ **–d** *dev* ] [ **–n***x* ] [ **–r** ] [ **–q** ]

**/usr/demo/jumpdemo** [ **–c** ] [ **–d** *dev* ] [ **–n***x* ] [ **–r** ] [ **–q** ]

**/usr/demo/spheresdemo** [ **–d** *dev* ] [ **–n***x* ] [ **–r** ] [ **–q** ]

## AVAILABILITY

These demos are available with the *Demos* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

### bouncedemo

**bouncedemo** displays a bouncing square.

### framedemo

**framedemo**

displays a series of frames, each of which contains a 256 by 256 image one-bit-deep pixels (that is, the image is a square monochrome bitmap, with 256 bits on a side). **framedemo** looks for the frames in the files **frame.1** through **frame.n** in the current working directory, and displays them in numerical order. A set of sample frames is available in the directory **/usr/demo/globeframes/***.

*Interactive Commands*

If you move the cursor onto the image surface, you can type certain commands to affect the rate at which the frames are displayed. The initial rate is one frame per second:

**f** Remove 1/20th of a second from the interval.

**F** Remove one second from the interval. **Ff** makes the interval as small as possible.

**s** Add 1/20th of a second.

**S** Add one second.

### jumpdemo

**jumpdemo** simulates the famous **Star Wars** jump to light-speed-sequence using vector drawing. Colored stars are drawn on color surfaces.

### spheresdemo

**spheresdemo** computes a random collection of shaded spheres. Colored spheres are drawn on color surfaces.

## OPTIONS

**–c** Rotate the color map to produce a sparkling effect.

**–d** *surface*

Run the demo on a surface other than the window or system console, for instance:

**bouncedemo –d /dev/cgone0**

**–n***x* Draw *x* items, or repeat a sequence *x* times.

**–r** Retain the window. This allows the image to reappear when uncovered instead of restarting the demo.

**–q** Quick exit. Useful for running several demos from within a shell script.

**NAME**

hack – replacement for rogue

**SYNOPSIS**

**hack** [ **–d** *hackdir* ] [ **–s all** | *player* ... ]

**DESCRIPTION**

**hack** is a display-oriented dungeons & dragons type game. Both display and command structure resemble *rogue*, although **hack** has twice as many monster types and requires three times as much memory.

Normally **hack** looks in **/usr/games/lib/hackdir** for the files listed below; this directory can be changed with the **–d** option. The **–s** option permits you to search the player record. Given the keyword **all**, **hack** lists all players; given the login name of a player, it lists all scores of that player.

**FILES**

| | |
|---|---|
| **record** | top 100 list (start with an empty file) |
| **news** | changes or bugs (start with no news file) |
| **data** | information about objects and monsters |
| **help** | introductory information (no doubt outdated) |
| **hh** | compacted version of help |
| **perm** | empty file used for locking |
| **rumors** | texts for fortune cookies |

**NAME**

hangman – computer version of the game hangman

**SYNOPSIS**

**/usr/games/hangman**

**DESCRIPTION**

In **hangman,** the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

**FILES**

**/usr/dict/words** on-line word list

## NAME

hunt – a multiplayer multiterminal game

## SYNOPSIS

**/usr/games/hunt**[**–m**] [ *hostname*] [ **–l** *name* ]

## DESCRIPTION

The object of the game **hunt** is to kill off the other players. There are no rooms, no treasures, and no monsters. Instead, you wander around a maze, find grenades, trip mines, and shoot down walls and players.

Your score is the ratio of number of kills to number of times you entered the game and is only kept for the duration of a single session of **hunt**. The more players you kill before you die, the better your score is.

**hunt** normally looks for an active game on the local network; if none is found, it starts one up on the local host. One may specify the location of the game by giving the *hostname* argument.

**hunt** only works on crt (vdt) terminals with at least 24 lines, 80 columns, and cursor addressing. The screen is divided in to 3 areas. On the right hand side is the status area. It shows you how much damage you've sustained, how many charges you have left, who's in the game, who's scanning (the asterisk in front of the name), who's cloaked (the plus sign in front of the name), and other players' scores. Most of the rest of the screen is taken up by your map of the maze, except for the 24th line, which is used for longer messages that do not fit in the status area.

**hunt** uses the same keys to move as **vi** does, for instance, **h**,**j**,**k**, and **l** for left, down, up, right respectively. To change which direction you're facing in the maze, use the upper case version of the movement key (for instance, HJKL).

Other commands are:

| | |
|---|---|
| f | Fire (in the direction you're facing) (Takes 1 charge) |
| g | Throw grenade (in the direction you're facing) (Takes 9 charges) |
| F | Throw satchel charge (Takes 25 charges) |
| G | Throw bomb (Takes 49 charges) |
| o | Throw small slime bomb (Takes 15 charges) |
| O | Throw big slime bomb (Takes 30 charges) |
| s | Scan (where other players are) (Takes 1 charge) |
| c | Cloak (where you are) (Takes 1 charge) |
| ^L | Redraw screen |
| q | Quit |

Knowing what the symbols on the screen often helps:

| | |
|---|---|
| –\|+ | Walls |
| /\ | Diagonal (deflecting) walls |
| # | Doors (dispersion walls) |
| ; | Small mine |
| g | Large mine |
| : | Shot |
| o | Grenade |
| O | Satchel charge |
| @ | Bomb |
| s | Small slime bomb |
| $ | Big slime bomb |
| ><^v | You facing right, left, up, or down |

`}{i!` Other players facing right, left, up, or down

`*` Explosion

```
 \|/
-*E-
 /|\
```
Grenade and large mine explosion

Satchel and bomb explosions are larger than grenades (5x5, 7x7, and 3x3 respectively).

Other helpful hints:

You can only fire in the direction you are facing.
You can only fire three shots in a row, then the gun must cool.
A shot only affects the square it hits.
Shots and grenades move 5 times faster than you do.
To stab someone,
    you must face that player and move at them.
Stabbing does 3 points worth of damage and shooting does 5 points.
You start with 15 charges and get 5 more for every new player.
A grenade affects the nine squares centered about the square it hits.
A satchel affects the twenty-five squares centered about the square it hits.
A bomb affects the forty-nine squares centered about the square it hits.
One small mine and one large mine is placed in the maze for every new player.
A mine has a 5% probability of tripping when you walk directly at it;
    50% when going sideways on to it; 95% when backing up on to it.
Tripping a mine costs you 5 points or 10 points respectively.
Defusing a mine is worth 1 charge or 9 charges respectively.
You cannot see behind you.
Scanning lasts for (20 times the number of players) turns.
    Scanning takes 1 ammo charge, so do not waste all your charges scanning.
You get 2 more damage capacity points and 2 damage points taken away
    whenever you kill someone.
Maximum typeahead is 5 characters.
A shot destroys normal (for instance, non-diagonal, non-door) walls.
Diagonal walls deflect shots and change orientation.
Doors disperse shots in random directions (up, down, left, right).
Diagonal walls and doors cannot be destroyed by direct shots but may
    be destroyed by an adjacent grenade explosion.
Walls regenerate, reappearing in the order they were destroyed.
    One percent of the regenerated walls will be diagonal walls or doors. When a wall is generated directly beneath a player, he is thrown in a random direction for a random period of time. When he lands, he sustains damage (up to 20 percent of the amount of damage he had before impact); that is, the less damage he had, the more nimble he is and therefore less likely to hurt himself on landing.

**ENVIRONMENT**

The environment variable **HUNT** is checked to get the player name. If you do not have this variable set, **hunt** will ask you what name you want to play under. You may also set up a single character keyboard map, but then you have to enumerate the options. For example:

**setenv HUNT "name=Sneaky,mapkey=zoFfGg1f2g3F4G"**

sets the player name to Sneaky, and the maps **z** to **o**, **F** to **f**, **G** to **g**, **1** to **f**, **2** to **g**, **3** to **F**, and **4** to **G**.

The *mapkey* option must be last.

It is a boring game if you are the only one playing.

**OPTIONS**

**–m** You enter the game as a monitor (you can see the action but you cannot play).

**–l** *name* Enter the game as player *name*.

**FILES**

**/usr/games/lib/hunt.driver** game coordinator

**LIMITATIONS**

**hunt** normally drives up the load average to be about (number_of_players + 0.5) greater than it would be without a **hunt** game executing. A limit of three players per host and nine players total is enforced by **hunt**.

**BUGS**

To keep up the pace, not everything is as realistic as possible.

**NAME**

life – John Conway's game of life

**SYNOPSIS**

**/usr/games/life**

**AVAILABILITY**

This game is available with the *Games* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**life** is a program that plays John Conway's game of life. It only runs under **sunview**(1).

When invoked, **life** will display a window with a small control panel at the top, and a large drawing area at the bottom. You can create pieces in the drawing area with the left button, and erase them with the middle button. When you select **Run** in the control panel, the pieces will begin to evolve, and the drawing region will update itself at a speed controlled by the slider labeled with **Fast** and **Slow**. **life** keeps track of all the pieces even if they are not visible. The scroll bars surrounding the drawing region can be used to see pieces that have moved out of view. There are some standard patterns that can be drawn by popping up a menu in the drawing subwindow.

The meaning of the items in the first row of the control panel (from left to right) are as follows. If you click on the picture which looks like a tic-tac-toe board, a grid will appear in the drawing region. If you click on **Step**, the mode will change from run mode (where the pieces update continuously) to step mode (where an update is only done when you click on **Step**). Following **Gen** is a number indicating the number of generations that have occured. The button marked **Find** will scroll so that at least one piece is in view. This is useful when all the pieces disappear from view. The button marked **Clear** will clear the drawing region, but leave the other controls unchanged. **Reset** will reset all the panel controls, but will not erase any of the pieces, and **Quit** Exits the tool. The second row contains two sliders. The first controls the update speed when in run mode, the second controls the size of the pieces.

**SEE ALSO**

**sunview**(1)

**NAME**

mille – play Mille Bornes

**SYNOPSIS**

**/usr/games/mille** [ file ]

**DESCRIPTION**

**mille** plays a two-handed game reminiscent of the Parker Brother's game of Mille Bornes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P Pick a card from the deck. This card is placed in the 'P' slot in your hand.

D Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a carriage-return or space. The carriage-return or space is required to allow recovery from typos which can be very expensive, like discarding safeties.

U Use a card. The card is again indicated by its number, followed by a carriage-return or space.

O Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.

Q Quit the game. This will ask for confirmation, just to be sure. Hitting DELETE (or RUBOUT) is equivalent.

S Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a RETURN character without a name, the save will be terminated and the game resumed.

R Redraw the screen from scratch. The command ^L (CTRL–L) will also work.

W Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save. (The game itself is a product of Parker Brothers, Inc.)

**SEE ALSO**

**curses**(3X)

**CARDS**

Here is some useful information. The number in brackets after the card name is the number of that card in the deck:

| Hazard | Repair | Safety |
|---|---|---|
| Out of Gas [2] | Gasoline [6] | Extra Tank [1] |
| Flat Tire [2] | Spare Tire [6] | Puncture Proof [1] |
| Accident [2] | Repairs [6] | Driving Ace [1] |
| Stop [4] | Go [14] | Right of Way [1] |
| Speed Limit [3] | End of Limit [6] | |

25 – [10], 50 – [10], 75 – [10], 100 – [12], 200 – [4]

RULES

**Object**: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

**Overview**: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. With the exception of the *speed limit* card, they can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

**Board Layout**: The board is split into several areas. From top to bottom, they are: **SAFETY AREA (unlabeled):**Thisiswherethesafeties played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

**Play**: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaches 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

**Hazard and Remedy Cards**: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastiness.

**Go** (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

**Stop** is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

**Speed Limit** is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

**End of Limit** is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

**Out of Gas** is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

**Flat Tire** is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.

**Accident** is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go*

card before they can play any more mileage.

**Safety Cards**: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn*.

**Right of Way** prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.

**Extra Tank** When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

**Puncture Proof** When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

**Driving Ace** When played, your opponent cannot play an *Accident* on your Battle Pile.

**Distance Cards**: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand*. A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action*.

**Coup Fouré**: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bornes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

**Scoring**: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

**Milestones Played**: Each player scores as many miles as they played before the trip ended.

**Each Safety**: 100 points for each safety in the Safety area.

**All 4 Safeties**: 300 points if all four safeties are played.

**Each Coup Fouré**: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

**Trip Completed**: 400 points bonus for completing the trip to 700 or 1000.

**Safe Trip**: 300 points bonus for completing the trip without using any 200 mile cards.

**Delayed Action**: 300 points bonus for finishing after the deck was exhausted.

**Extension**: 200 points bonus for completing a 1000 mile trip.

**Shut-Out**: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

## NAME

monop – Monopoly game

## SYNOPSIS

**/usr/games/monop** [*filename*]

## DESCRIPTION

**monop** is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", that is, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, for instance a name, place or person, you can type **?** to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

## USAGE

### Commands

**quit:** Quit game. This allows you to quit the game. It asks you if you are sure.

**print** Print board. This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

| | |
|---|---|
| Name | The first ten characters of the name of the square |
| Own | The *number* of the owner of the property. |
| Price | The cost of the property (if any) |
| Mg | This field has a '*' in it if the property is mortgaged |
| # | If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it. |
| Rent | Current rent on the property. If it is not owned, there is no rent. |

**where:** where players are: Tells you where all the players are. A '*' indicates the current player.

**own holdings:**
List your own holdings, that is, money, get-out-of-jail-free cards, and property.

**holdings:**
Holdings list. Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type **done**.

**shell:** Shell escape. Escape to a shell. When the shell dies, the program continues where you left off.

**mortgage:**
Mortgage property. Sets up a list of mortgageable property, and asks which you wish to mortgage.

**unmortgage:**
Unmortgage property. Unmortgage mortgaged property.

**buy:** Buy houses. Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

**sell:** Sell houses. Sets up a list of monopolies from which you can sell houses. it operates in an

analogous manner to **buy**

**card:** Card for jail. Use a get-out-of-jail-free card to get out of jail. If you are not in jail, or you do not have one, it tells you so.

**pay:** Pay for jail. Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you are not there.

**trade:** This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

**resign:** Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

**save:** Save game. Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the **monop** command, or by using the **restore** command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

**restore:**
Restore game. Read in a previously saved game from a file. It leaves the file intact.

**roll:** Roll the dice and move forward to your new location. If you simply hit the RETURN key instead of a command, it is the same as typing *roll*.

**FILES**

**/usr/games/lib/cards.pck** chance and community chest cards

**BUGS**

No command can be given an argument instead of a response to a query.

**NAME**

moo – guessing game

**SYNOPSIS**

**/usr/games/moo**

**DESCRIPTION**

**moo** is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A "cow" is a correct digit in an incorrect position. A "bull" is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).

**NAME**

number – convert Arabic numerals to English

**SYNOPSIS**

**/usr/games/number**

**DESCRIPTION**

**number** copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

**NAME**

primes – print all primes larger than some given number

**SYNOPSIS**

**/usr/games/primes** [ *number* ]

**DESCRIPTION**

**primes** reads a number from the standard input and prints all primes larger than the given number. If *number* is given as an argument, it uses that number rather than reading one from the standard input.

**BUGS**

It obviously cannot print *all* primes larger than some given number. It will not behave very sensibly when it overflows an **int**.

**NAME**

quiz – test your knowledge

**SYNOPSIS**

**/usr/games/quiz** [ –i*filename* ] [ –**t** ] [ *category1 category2* ]

**DESCRIPTION**

**quiz** gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, **quiz** gives instructions and lists the available categories.

**quiz** tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, **quiz** reports a score and terminates.

The –**t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The –**i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line     = category newline | category ':' line
category = alternate | category 'l' alternate
alternate = empty | alternate primary
primary  = character | '[' category ']' | option
option   = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with **sh**(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, **quiz** will refrain from asking it.

**FILES**

**/usr/games/quiz.k/***

**BUGS**

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

**NAME**

rain – animated raindrops display

**SYNOPSIS**

**/usr/games/rain**

**DESCRIPTION**

**rain**'s display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use **termcap**, the TERM environment variable must be set (and exported) to the type of the terminal being used.

**FILES**

**/etc/termcap**

**NAME**

random – select lines randomly from a file

**SYNOPSIS**

**/usr/games/random** [ **–er** ] [ *divisor* ]

**DESCRIPTION**

**random** acts as a text filter, randomly selecting lines from its standard input to write to the standard output. The probability that a given line is selected is normally 1/2; if a *divisor* is specified, it is treated as a floating-point number, and the probability is 1/*divisor* instead.

**OPTIONS**

**–e** Don't read the standard input or write to the standard output. Instead, exit with a random exit status between 0 and 1, or between 0 and *divisor*-1 if *divisor* is specified.

**–r** Don't buffer the output. If **–r** is not used, output is buffered in blocks, or line-buffered if the standard output is a terminal.

**NAME**

robots – fight off villainous robots

**SYNOPSIS**

**/usr/games/robots** [ **–sjta** ] [ *scorefile* ]

**DESCRIPTION**

**robots** pits you against evil robots, who are trying to kill you (which is why they are evil). Fortunately for you, even though they are evil, they are not very bright and have a habit of bumping into each other, thus destroying themselves. In order to survive, you must get them to kill each other off, since you have no offensive weaponry.

Since you are stuck without offensive weaponry, you are endowed with one piece of defensive weaponry: a teleportation device. When two robots run into each other or a junk pile, they die. If a robot runs into you, you die. When a robot dies, you get 10 points, and when all the robots die, you start on the next field. This keeps up until they finally get you.

Robots are represented on the screen by a '+', the junk heaps from their collisions by a '*', and you (the good guy) by a '@'.

The commands are:

**h** move one square left

**l** move one square right

**k** move one square up

**j** move one square down

**y** move one square up and left

**u** move one square up and right

**b** move one square down and left

**n** move one square down and right

**.** (also space) do nothing for one turn

**HJKLBNYU**
run as far as possible in the given direction

**>** do nothing for as long as possible

**t** teleport to a random location

**w** wait until you die or they all do

**q** quit

**^L** redraw the screen

All commands can be preceded by a count.

If you use the 'w' command and survive to the next level, you will get a bonus of 10% for each robot which died after you decided to wait. If you die, however, you get nothing. For all other commands, the program will save you from typos by stopping short of being eaten. However, with 'w' you take the risk of dying by miscalculation.

Only five scores are allowed per user on the score file. If you make it into the score file, you will be shown the list at the end of the game. If an alternate score file is specified, that will be used instead of the standard file for scores.

**OPTIONS**

**–s** Do not play, just show the score file.

**–j** Jump, when you run, don't show any intermediate positions; only show things at the end. This is useful on slow terminals.

**–t** Teleport automatically when you have no other option. This is a little disconcerting until you get used to it, and then it is very nice.

**–a** Advance into the higher levels directly, skipping the lower, easier levels.

**FILES**

**/usr/games/lib/robots_roll** the score file

**BUGS**

Bugs? You *crazy*, man?!?

## NAME

snake, snscore – display chase game

## SYNOPSIS

**/usr/games/snake** [ **–w***n* ] [ **–l***n* ]
**/usr/games/snscore**

## DESCRIPTION

**snake** is a display-based game which must be played on a CRT terminal from among those supported by **vi**(1). The object of the game is to make as much money as possible without getting eaten by the snake. The **–l** and **–w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as **vi**(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

**sefc** These keys are like hjkl but form a directed pad around the d key.

**HJKL** These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.

**SEFC** Likewise for the upper case versions on the left.

**ATPB** These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, for example, P is at the far right of the keyboard.

**x** This lets you quit the game at any time.

**p** Points in a direction you might want to go.

**w** Space warp to get out of tight squeezes, at a price.

**!** Shell escape

**^Z** Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run **/usr/games/snscore .**

## FILES

| | |
|---|---|
| **/usr/games/lib/snakerawscores** | database of personal bests |
| **/usr/games/lib/snake.log** | log of games played |

## BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

## NAME

sunview_demos, canvas_demo, cursor_demo – Window-System demonstration programs

## SYNOPSIS

**/usr/demo/canvas_demo**

**/usr/demo/cursor_demo**

## AVAILABILITY

These demos are available with the *SunView 1 Demos* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

### canvas_Demo

**canvas_demo** demonstrates the capabilities of the canvas subwindow package. It consists of two subwindows: a control panel and a canvas. By adjusting the items on the control panel, you can manipulate the attributes of the canvas, and see the results.

### cursor_Demo

**cursor_demo** demonstrates what you can do with cursors. A single control panel is provide for adjusting the various cursor attributes. As you adjust the items on the control panel, the panel's cursor changes in appearance.

## NAME

trek – trekkie game

## SYNOPSIS

**/usr/games/trek** [ [ **–a** ] *filename* ]

## DESCRIPTION

**trek** is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **–a** flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commodore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

## COMMAND SUMMARY

**abandon**
**c**loak **u**p/**d**own
computer request; ...
**destruct**
**help**
**l**rscan
**p**hasers **a**utomatic amount
**p**hasers **m**annual amt1 course1 spread1 ...
**t**orpedo course [yes] angle/**n**o
**ram** course distance
**shell**
srscan [yes/**n**o]
**s**tatus
**u**ndock
warp warp_factor

**c**apture

**d**amages
**d**ock
**i**mpulse course distance
**m**ove course distance

**r**est time
**s**hields **u**p/**d**own

**terminate** yes/no
visual course

**NAME**

worm – play the growing worm game

**SYNOPSIS**

**/usr/games/worm** [ *size* ]

**DESCRIPTION**

In **worm,** you are a little worm, your body is the o 's on the screen and your head is the @ . You move with the hjkl keys (as in the game **snake**). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit; if your worm eats the digit it will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

**BUGS**

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

**NAME**

worms – animate worms on a display terminal

**SYNOPSIS**

**/usr/games/worms** [ **–field** ] [ **–length #** ] [ **–number #** ] [ **–trail** ]

**DESCRIPTION**

**–field** makes a "field" for the worm(s) to eat; **–trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

**FILES**

**/etc/termcap**

**SEE ALSO**

*Snails* by Karl Heuer

**BUGS**

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

**NAME**

wump – the game of hunt-the-wumpus

**SYNOPSIS**

**/usr/games/wump**

**DESCRIPTION**

**wump** plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

## NAME

miscellaneous – miscellaneous useful information pages

## DESCRIPTION

This section contains miscellaneous documentation, mostly in the area of text processing macro packages for *troff*(1).

## LIST OF MISC. TABLES

| Name | Appears on Page | Description |
|---|---|---|
| **ascii** | **ascii**(7) | map of ASCII character set |
| **eqnchar** | **eqnchar**(7) | special character definitions for eqn |
| **filesystem** | **filesystem**(7) | filesystem organization |
| **hier** | **hier**(7) | file system hierarchy |
| **man** | **man**(7) | macros to format Reference Manual pages |
| **me** | **me**(7) | macros for formatting papers |
| **ms** | **ms**(7) | text formatting macros |

**NAME**

ascii – map of ASCII character set

**SYNOPSIS**

**cat /usr/pub/ascii**

**DESCRIPTION**

**/usr/pub/ascii** is a map of the ASCII character set, to be printed as needed. It contains octal and hexadecimal values for each character. While not included in that file, a chart of decimal values is also shown here.

*Octal — Character*

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | NUL | 001 | SOH | 002 | STX | 003 | ETX | 004 | EOT | 005 | ENQ | 006 | ACK | 007 | BEL |
| 010 | BS | 011 | HT | 012 | NL | 013 | VT | 014 | NP | 015 | CR | 016 | SO | 017 | SI |
| 020 | DLE | 021 | DC1 | 022 | DC2 | 023 | DC3 | 024 | DC4 | 025 | NAK | 026 | SYN | 027 | ETB |
| 030 | CAN | 031 | EM | 032 | SUB | 033 | ESC | 034 | FS | 035 | GS | 036 | RS | 037 | US |
| 040 | SP | 041 | ! | 042 | " | 043 | # | 044 | $ | 045 | % | 046 | & | 047 | ´ |
| 050 | ( | 051 | ) | 052 | * | 053 | + | 054 | , | 055 | – | 056 | . | 057 | / |
| 060 | 0 | 061 | 1 | 062 | 2 | 063 | 3 | 064 | 4 | 065 | 5 | 066 | 6 | 067 | 7 |
| 070 | 8 | 071 | 9 | 072 | : | 073 | ; | 074 | < | 075 | = | 076 | > | 077 | ? |
| 100 | @ | 101 | A | 102 | B | 103 | C | 104 | D | 105 | E | 106 | F | 107 | G |
| 110 | H | 111 | I | 112 | J | 113 | K | 114 | L | 115 | M | 116 | N | 117 | O |
| 120 | P | 121 | Q | 122 | R | 123 | S | 124 | T | 125 | U | 126 | V | 127 | W |
| 130 | X | 131 | Y | 132 | Z | 133 | [ | 134 | \ | 135 | ] | 136 | ^ | 137 | _ |
| 140 | ` | 141 | a | 142 | b | 143 | c | 144 | d | 145 | e | 146 | f | 147 | g |
| 150 | h | 151 | i | 152 | j | 153 | k | 154 | l | 155 | m | 156 | n | 157 | o |
| 160 | p | 161 | q | 162 | r | 163 | s | 164 | t | 165 | u | 166 | v | 167 | w |
| 170 | x | 171 | y | 172 | z | 173 | { | 174 | \| | 175 | } | 176 | ~ | 177 | DEL |

*Hexadecimal — Character*

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | NUL | 01 | SOH | 02 | STX | 03 | ETX | 04 | EOT | 05 | ENQ | 06 | ACK | 07 | BEL |
| 08 | BS | 09 | HT | 0A | NL | 0B | VT | 0C | NP | 0D | CR | 0E | SO | 0F | SI |
| 10 | DLE | 11 | DC1 | 12 | DC2 | 13 | DC3 | 14 | DC4 | 15 | NAK | 16 | SYN | 17 | ETB |
| 18 | CAN | 19 | EM | 1A | SUB | 1B | ESC | 1C | FS | 1D | GS | 1E | RS | 1F | US |
| 20 | SP | 21 | ! | 22 | " | 23 | # | 24 | $ | 25 | % | 26 | & | 27 | ´ |
| 28 | ( | 29 | ) | 2A | * | 2B | + | 2C | , | 2D | – | 2E | . | 2F | / |
| 30 | 0 | 31 | 1 | 32 | 2 | 33 | 3 | 34 | 4 | 35 | 5 | 36 | 6 | 37 | 7 |
| 38 | 8 | 39 | 9 | 3A | : | 3B | ; | 3C | < | 3D | = | 3E | > | 3F | ? |
| 40 | @ | 41 | A | 42 | B | 43 | C | 44 | D | 45 | E | 46 | F | 47 | G |
| 48 | H | 49 | I | 4A | J | 4B | K | 4C | L | 4D | M | 4E | N | 4F | O |
| 50 | P | 51 | Q | 52 | R | 53 | S | 54 | T | 55 | U | 56 | V | 57 | W |
| 58 | X | 59 | Y | 5A | Z | 5B | [ | 5C | \ | 5D | ] | 5E | ^ | 5F | _ |
| 60 | ` | 61 | a | 62 | b | 63 | c | 64 | d | 65 | e | 66 | f | 67 | g |
| 68 | h | 69 | i | 6A | j | 6B | k | 6C | l | 6D | m | 6E | n | 6F | o |
| 70 | p | 71 | q | 72 | r | 73 | s | 74 | t | 75 | u | 76 | v | 77 | w |
| 78 | x | 79 | y | 7A | z | 7B | { | 7C | \| | 7D | } | 7E | ~ | 7F | DEL |

*Decimal — Character*

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | 1 | SOH | 2 | STX | 3 | ETX | 4 | EOT | 5 | ENQ | 6 | ACK | 7 | BEL |
| 8 | BS | 9 | HT | 10 | NL | 11 | VT | 12 | NP | 13 | CR | 14 | SO | 15 | SI |
| 16 | DLE | 17 | DC1 | 18 | DC2 | 19 | DC3 | 20 | DC4 | 21 | NAK | 22 | SYN | 23 | ETB |
| 24 | CAN | 25 | EM | 26 | SUB | 27 | ESC | 28 | FS | 29 | GS | 30 | RS | 31 | US |
| 32 | SP | 33 | ! | 34 | " | 35 | # | 36 | $ | 37 | % | 38 | & | 39 | ´ |
| 40 | ( | 41 | ) | 42 | * | 43 | + | 44 | , | 45 | − | 46 | . | 47 | / |
| 48 | 0 | 49 | 1 | 50 | 2 | 51 | 3 | 52 | 4 | 53 | 5 | 54 | 6 | 55 | 7 |
| 56 | 8 | 57 | 9 | 58 | : | 59 | ; | 60 | < | 61 | = | 62 | > | 63 | ? |
| 64 | @ | 65 | A | 66 | B | 67 | C | 68 | D | 69 | E | 70 | F | 71 | G |
| 72 | H | 73 | I | 74 | J | 75 | K | 76 | L | 77 | M | 78 | N | 79 | O |
| 80 | P | 81 | Q | 82 | R | 83 | S | 84 | T | 85 | U | 86 | V | 87 | W |
| 88 | X | 89 | Y | 90 | Z | 91 | [ | 92 | \ | 93 | ] | 94 | ^ | 95 | _ |
| 96 | ` | 97 | a | 98 | b | 99 | c | 100 | d | 101 | e | 102 | f | 103 | g |
| 104 | h | 105 | i | 106 | j | 107 | k | 108 | l | 109 | m | 110 | n | 111 | o |
| 112 | p | 113 | q | 114 | r | 115 | s | 116 | t | 117 | u | 118 | v | 119 | w |
| 120 | x | 121 | y | 122 | z | 123 | { | 124 | \| | 125 | } | 126 | ~ | 127 | DEL |

**FILES**

**/usr/pub/ascii** Online chart of octal and hexadecimal values for the ASCII character set.

## NAME

eqnchar – special character definitions for eqn

## SYNOPSIS

**eqn /usr/pub/eqnchar** [ *filename* ] | **troff** [ *options* ]

**neqn /usr/pub/eqnchar** [ *filename* ] | **nroff** [ *options* ]

## DESCRIPTION

**eqnchar** contains **troff**(1) and **nroff**(1) character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with **eqn**(1) and **neqn**(1). It contains definitions for the following characters

| | | | | | |
|---|---|---|---|---|---|
| *ciplus* | ⊕ | *//* | // | *square* | □ |
| *citimes* | ⊗ | *langle* | 〈 | *circle* | ○ |
| *wig* | ~ | *rangle* | 〉 | *blot* | □ |
| *-wig* | ≈ | *hbar* | ħ | *bullet* | • |
| *>wig* | ≳ | *ppd* | ⊥ | *prop* | ∝ |
| *<wig* | ≲ | *<->* | ↔ | *empty* | ∅ |
| *=wig* | ≅ | *<=>* | ⇔ | *member* | ∈ |
| *star* | ⋆ | */<* | ≮ | *nomem* | ∉ |
| *bigstar* | ✳ | */>* | ≯ | *cup* | ∪ |
| *=dot* | ≐ | *ang* | ∠ | *cap* | ∩ |
| *orsign* | ∨ | *rang* | ⦥ | *incl* | ⊏ |
| *andsign* | ∧ | *3dot* | ⋮ | *subset* | ⊂ |
| *=del* | ≙ | *thf* | ∴ | *supset* | ⊃ |
| *oppA* | ∀ | *quarter* | ¼ | *!subset* | ⊆ |
| *oppE* | ∃ | *3quarter* | ¾ | *!supset* | ⊇ |
| *angstrom* | Å | *degree* | ° | | |

## FILES

**/usr/pub/eqnchar**

## SEE ALSO

**eqn**(1), **neqn**(1), **nroff**(1), **troff**(1)

## NAME

filesystem – file system organization

## SYNOPSIS

/

**/usr**

## DESCRIPTION

The SunOS file system tree is organized for easy administration. Distinct areas within the file system tree are provided for files that are private to one machine, files that can be shared by multiple machines of a common architecture, files that can be shared by all machines, and home directories. This organization allows the sharable files to be stored on one machine, while being accessed by many machines using a remote file access mechanism such as Sun's Network File System (NFS). Grouping together similar files makes the file system tree easier to upgrade and manage.

The file system tree consists of a root file system and a collection of mountable file systems. The **mount**(8) program attaches mountable file systems to the file system tree at mount points (directory entries) in the root file system, or other previously mounted file systems. Two file systems, / (the root) and **/usr**, must be mounted in order to have a fully functional system. The root file system is mounted automatically by the kernel at boot time; the **/usr** file system is mounted by the **/etc/rc.boot** script, which is run as part of the booting process.

The root file system contains files that are unique to each machine; it can not be shared among machines. The root file system contains the following directories:

**/dev** Character and block special files. Device files provide hooks into hardware devices or operating system facilities. The **MAKEDEV**(8) command builds device files in the **/dev** directory. Typically, device files are built to match the kernel and hardware configuration of the machine.

**/etc** Various configuration files and system administration databases that are machine specific. You can think of **/etc** as the "home directory" of a machine, defining its "identity." Executable programs are no longer kept in **/etc**.

**/home** Mount points for home directories. This directory may be arranged so that shared user files are placed under the directory **/home/***machine-name* on machines serving as file servers. Machines may then be locally configured with mount points under **/home** for all of the file servers of interest, with the name of the mount point being the name of the file server.

**/mnt** A generic mount point. This is an empty directory available for temporarily mounting file systems on.

**/sbin** Executable programs that are needed in the boot process before **/usr** is mounted. **/sbin** contains *only* those programs that are needed in order to mount the **/usr** file system: **hostname**(1), **ifconfig**(8C), **init**(8), **mount**(8), and **sh**(1). After **/usr** is mounted, the full complement of utilities are available.

**/tmp** Temporary files that are deleted at reboot time.

**/var** Files, such as log files, that are unique to a machine but that can grow to an arbitrary ("variable") size.

**/var/adm** System logging and accounting files.

**/var/preserve**
Backup files for **vi**(1) and **ex**(1).

**/var/spool** Subdirectories for files used in printer spooling, mail delivery, **cron**(8), **at**(1), etc.

**/var/tmp** Transitory files that are not deleted at reboot time.

Because it is desirable to keep the root file system small, larger file systems are often mounted on **/var** and **/tmp**.

The file system mounted on **/usr** contains architecture-dependent and architecture-independent shareable files. The subtree rooted at **/usr/share** contains architecture-independent shareable files; the rest of the **/usr** tree contains architecture-dependent files. By mounting a common remote file system, a group of machines with a common architecture may share a single **/usr** file system. A single **/usr/share** file system can be shared by machines of any architecture. A machine acting as a file server may export many different **/usr** file systems to support several different architectures and operating system releases. Clients usually mount **/usr** read-only to prevent their accidentally modifying any shared files. The **/usr** file system contains the following subdirectories:

| | |
|---|---|
| **/usr/5bin** | System V executables. |
| **/usr/5include** | System V include files. |
| **/usr/5lib** | System V library files. |
| **/usr/bin** | Executable programs. The bulk of the system utilities are located here. |
| **/usr/dict** | Dictionary databases. |
| **/usr/etc** | Executable system administration programs. |
| **/usr/games** | Executable game programs and data. |
| **/usr/include** | Include files. |
| **/usr/lib** | Program libraries and various architecture-dependent databases. |
| **/usr/pub** | Various data files. |
| **/usr/ucb** | Executable programs descended from the Berkeley Software Distribution. |
| **/usr/share** | Subtree for architecture-independent shareable files. |
| **/usr/share/man** | Subdirectories for the on-line reference manual pages. |
| **/usr/share/lib** | Architecture-independent databases. |

A machine with disks may export root file systems, swap files and **/usr** file systems to diskless or partially-disked machines, which mount these into the standard file system hierarchy. The standard directory tree for exporting these file systems is:

| | |
|---|---|
| **/export** | The root of the exported file system tree. |
| **/export/exec/***architecture-name* | The exported **/usr** file system supporting *architecture-name* for the current release. |
| **/export/exec/***architecture-name***.***release-name* | The exported **/usr** file system supporting *architecture-name* for SunOS *release-name*. |
| **/export/share** | The exported common **/usr/share** directory tree. |
| **/export/root/***hostname* | The exported root file system for *hostname*. |
| **/export/swap/***hostname* | The exported swap file for *hostname*. |
| **/export/var/***hostname* | The exported **/var** directory tree for *hostname*. |
| **/export/dump/***hostname* | The exported dump file for *hostname*. |
| **/export/crash/***hostname* | The exported crash dump directory for *hostname*. |

**Changes from Previous Releases**

The file system layout described here is quite a bit different from the layout employed previous to release 4.0 of SunOS. For compatibility with earlier releases of SunOS, and other versions of the UNIX system, symbolic links are provided for various files and directories linking their previous names to their current locations. The symbolic links provided include:

| | |
|---|---|
| **/bin —> /usr/bin** | All programs previously located in **/bin** are now in **/usr/bin**. |
| **/lib —> /usr/lib** | All files previously located in **/lib** are now in **/usr/lib**. |
| **/usr/adm —> /var/adm** | The entire **/usr/adm** directory has been moved to **/var/adm**. |
| **/usr/spool —> /var/spool** | The entire **/usr/spool** directory has been moved to **/var/spool**. |
| **/usr/tmp —> /var/tmp** | The **/usr/tmp** directory has been moved to **/var/tmp**. |

**/etc/termcap —> /usr/share/lib/termcap**

**/usr/5lib/terminfo —> /usr/share/lib/terminfo**

**/usr/lib/me —> /usr/share/lib/me**

**/usr/lib/ms —> /usr/share/lib/ms**

**/usr/lib/tmac —> /usr/share/lib/tmac**

**/usr/man —> /usr/share/man**

The following program binaries have been moved from **/etc** to **/usr/etc** with symbolic links to them left in **/etc**: **arp, clri, cron, chown, chroot, config, dkinfo, dmesg, dump, fastboot, fasthalt, fsck, halt, ifconfig, link, mkfs, mknod, mount, ncheck, newfs, pstat, rdump, reboot, renice, restore, rmt, rrestore, shutdown, umount, update, unlink,** and **vipw**.

In addition, some files and directories have been moved with no symbolic link left behind in the old location:

| *Old Name* | *New Name* |
|---|---|
| **/etc/biod** | **/usr/etc/biod** |
| **/etc/fsirand** | **/usr/etc/fsirand** |
| **/etc/getty** | **/usr/etc/getty** |
| **/etc/in.rlogind** | **/usr/etc/in.rlogind** |
| **/etc/in.routed** | **/usr/etc/in.routed** |
| **/etc/in.rshd** | **/usr/etc/in.rshd** |
| **/etc/inetd** | **/usr/etc/inetd** |
| **/etc/init** | **/usr/etc/init** |
| **/etc/nfsd** | **/usr/etc/nfsd** |
| **/etc/portmap** | **/usr/etc/portmap** |
| **/etc/rpc.lockd** | **/usr/etc/rpc.lockd** |
| **/etc/rpc.statd** | **/usr/etc/rpc.statd** |
| **/etc/ypbind** | **/usr/etc/ypbind** |
| **/usr/lib/sendmail.cf** | **/etc/sendmail.cf** |
| **/usr/preserve** | **/var/preserve** |
| **/usr/lib/aliases** | **/etc/aliases** |
| **/stand** | **/usr/stand** |
| **/etc/yp** | **/var/yp** |

Note: with this new file system organization, the approach to repairing a broken file system changes. One must mount **/usr** before doing an **fsck**(8), for example. If the mount point for **/usr** has been destroyed, **/usr** can be mounted temporarily on **/mnt** or **/tmp**. If the root file system on a standalone system is so badly damaged that none of these mount points exist, or if **/sbin/mount** has been corrupted, the only way to repair it may be to re-install the root file system.

**SEE ALSO**

**at**(1), **ex**(1), **hostname**(1), **sh**(1), **vi**(1), **intro**(4), **nfs**(4P), **hier**(7), **ifconfig**(8C), **init**(8), **MAKEDEV**(8), **mount**(8), **fsck**(8), **rc**(8)

**NAME**

hier – file system hierarchy

**DESCRIPTION**

The following outline gives a quick tour through a typical SunOS file system hierarchy:

/ root directory of the file system

/**dev**/ devices (Section 4)

**MAKEDEV**
shell script to create special files

**MAKEDEV.local**
site specific part of **MAKEDEV**

**console** main system console, **console**(4S)

**drum** paging device, **drum**(4)

***mem** memory special files, **mem**(4S)

**null** null file or data sink, **null**(4)

**pty[p-z]***
pseudo terminal controllers, **pty**(4)

**tty[ab]** CPU serial ports, **zs**(4S)

**tty[0123][0-f]**
MTI serial ports **mti**(4S)

**tty[hijk][0-f]**
ALM-2 serial ports **mcp**(4S)

**tty[p-z]***
pseudo terminals, **pty**(4)

**vme*** VME bus special files, **mem**(4S)

**win** window system special files, **win**(4S)

**xy*** disks, **xy**(4S)

**rxy*** raw disk interfaces, **xy**(4S)

...

/**etc**/ system-specific maintenance and data files

**dumpdates**
dump history, **dump**(8)

**exports** table of file systems exportable with NFS, **exports**(5)

**fstab** file system configuration table, **fstab**(5)

**group** group file, **group**(5)

**hosts** host name to network address mapping file, **hosts**(5)

**hosts.equiv**
list of trusted systems, **hosts.equiv**(5)

**motd** message of the day, **login**(1)

**mtab** mounted file table, **mtab**(5)

**networks**
network name to network number mapping file, **networks**(5)

**passwd** password file, **passwd**(5)

**phones** private phone numbers for remote hosts, as described in **phones**(5)

**printcap**
table of printers and capabilities, **printcap**(5)

**protocols**
protocol name to protocol number mapping file, **protocols**(5)

**rc** shell program to bring the system up multiuser

**rc.boot** startup file run at boot time

**rc.local** site dependent portion of **rc**

**remote** names and description of remote hosts for **tip**(1C), **remote**(5)

**services**
network services definition file, **services**(5)

**ttytab** database of terminal information used by **getty**(8)
...
**/export/**
directory of exported files and file systems for clients, including swap files, root, and **/usr** file systems
**/home/** directory of mount points for remote-mounted home directories and shared file systems
*user* home (initial working) directory for *user*
**.profile** set environment for **sh**(1), **environ**(5V)
**.project**
what you are doing (used by (**finger**(1))
**.cshrc** startup file for **csh**(1)
**.exrc** startup file for **ex**(1)
**.plan** what your short-term plans are (used by **finger**(1))
**.rhosts** host equivalence file for **rlogin**(1C)
**.mailrc** startup file for **mail**(1)
**calendar**
user's datebook for **calendar**(1)
...
**/lost+found**
directory for connecting detached files for **fsck**(8)
**/mnt/** mount point for file systems mounted temporarily
**/sbin/** executable programs needed to mount **/usr/**
**hostname**
**ifconfig**
**init**
**mount**
**sh**
**/tmp/** temporary files, usually on a fast device, see also **/var/tmp/**
**ctm*** used by **cc**(1V)
**e*** used by **ed**(1)
...
**/var/** directory of files that tend to grow or vary in size
**adm/** administrative log files
**lastlog** record of recent logins, **utmp**(5)
**lpacct** line printer accounting **lpr**(1)
**messages**
system messages
**tracct** phototypesetter accounting, **troff**(1)
**utmp** table of currently logged in users, **utmp**(5)
**vaacct, vpacct**
varian and versatec accounting **vtroff**(1), **pac**(8)
**wtmp** login history, **utmp**(5)
...
**preserve/**
editor temporaries preserved here after crashes/hangups
**spool/** delayed execution files
**cron/** used by **cron**(8)
**lpd/** used by **lpr**(1)
**lock** present when line printer is active
**cf*** copy of file to be printed, if necessary
**df*** control file for print job
**tf*** transient control file, while *lpr* is working
**mail/** mailboxes for **mail**(1)

*name* mail file for user *name*
*name*.**lock**
lock file while *name* is receiving mail
**mqueue/**
mail queue for **sendmail**(8)
**secretmail/**
like **mail/**, but used by **xsend**(1)
**uucp/** work files and staging area for **uucp**(1C)
**LOGFILE**
summary log
**LOG.*** log file for one transaction
...
**tmp/** temporary files, to keep **/tmp/** small
**raster** used by **plot**(1G)
**stm*** used by **sort**(1V)
...
**yp/** Yellow Pages database files, **ypfiles**(5)
**/usr/** general-purpose directory, usually a mounted file system
**bin/** utility programs
**as** assembler, **as**(1)
**cc** C compiler executive, c.f. **/usr/lib/ccom**, **/usr/lib/cpp**, **/usr/lib/c2**
**csh** the C-shell, **csh**(1)
**sh** the Bourne shell, **sh**(1)
...
**demo/** demonstration programs
**diag/** system tests and diagnostics
**dict/** word lists, etc.
**spellhist**
history file for **spell**(1)
**words** principal word list, used by **look**(1)
...
**etc/** system administration programs; c.f. section 8
**catman** update preformatted man pages, **catman**(8)
**cron** the clock daemon, **cron**(8)
**dump** file system backup program **dump**(8)
**getty** part of **login**(1), **getty**(8)
**in.comsat**
biff server (incoming mail daemon), **comsat**(8C)
**init** the parent of all processes, **init**(8)
**mount** **mount**(8)
**yp/** Yellow Pages programs
**ypinit** build and install Yellow Pages database, **ypinit**(8)
**yppush** force propagation of a changed Yellow Pages map, **yppush**(8)
**ypset** point ypbind at a particular server, **ypset**(8)
...
...
**games/**
**backgammon**
**lib/** library directory for game scores, etc.
**quiz.k/** what **quiz**(6) knows
**africa** countries and capitals
**index** category index
...

```
                ...
        ...
hosts/  symbolic links to rsh(1C) for commonly accessed remote hosts
include/
        standard #include files
        a.out.h  object file layout, a.out(5)
        images/  icon images
        machine/
                header files from /usr/share/sys/sys/machine; may be a symbolic link
        math.h   intro(3M)
        net/     header files from /usr/share/sys/sys/net; may be a symbolic link
        nfs/     header files used in the Network File System (NFS)
        stdio.h  standard I/O, intro(3S)
        sys/     kernel header files, c.f. /usr/share/sys/sys
        ...
lib/    object libraries, compiler program binaries, and other data
        ccom     C compiler proper
        cpp      C preprocessor
        c2       C code improver
        eign     list of English words to be ignored by ptx(1)
        font/    fonts for troff(1)
                 ftR      Times Roman
                 ftB      Times Bold
                 ...
        libc.a   system calls, standard I/O, etc. (2,3,3S)
        libm.a   math library, intro(3M)
        lint/    utility files for lint
                 lint[12] subprocesses for lint(1V)
                 llib-lc  dummy declarations for /usr/lib/libc.a, used by lint(1V)
                 llib-lm  dummy declarations for /usr/lib/libm.a
                 ...
        units    conversion tables for units(1)
        uucp/    programs and data for uucp(1C)
                 L.sys    remote system names and numbers
                 uucico   the real copy program
                 ...
        ...
local/  locally maintained software
old/    obsolete and unsupported programs
pub/    publicly readable data files
sccs/   binaries of programs that compose the source code control system (SCCS)
src/    system source code tree
stand/  standalone programs (not run under the Sun Operating System)
share/  architecture independent files
        lib/     architecture independent data files
                 termcap
                         description of terminal capabilities, termcap(5)
                 tmac/    macros for troff(1)
                          tmac.an
                                  macros for man(7)
                          tmac.s   macros for ms(7)
                          ...
                 ...
```

**man/** on-line reference manual pages, **man**(1)
**man?/** source files (**nroff**(1)) for sections 1 through 8 of the manual
**as.1**
...
**cat?/** preformatted pages for sections 1 through 8 of the manual
...
**sys/** SunOS kernel source and object modules
**ucb/** binaries of programs developed at the University of California, Berkeley
**ex** line-oriented editor for experienced users, **ex**(1)
**vi** screen-oriented editor, **vi**(1)
...

**/vmunix**
the SunOS kernel binary

## SEE ALSO

**filesystem**(7), **find**(1), **finger**(1), **grep**(1V), **ls**(1V), **rlogin**(1C), **whatis**(1), **whereis**(1), **which**(1), **ncheck**(8)

## BUGS

The locations of files are subject to change without notice; the organization of your file system may vary.

This list is incomplete.

## NAME

man – macros to format Reference Manual pages

## SYNOPSIS

**nroff –man** *filename...*

**troff –man** *filename...*

## DESCRIPTION

These macros are used to lay out the reference pages in this manual.

Any text argument *t* may be zero to six words. Quotes may be used to include blanks in a "word". If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way **.I** may be used to italicize a whole line, or **.SB** may be used to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **–man**:

*R '®', '(Reg)' in **nroff.**

*S Change to default type size.

**Requests**

| *Request* | *Cause Break* | *If no Argument* | *Explanation* |
|---|---|---|---|
| **.B** *t* | no | *t*=n.t.l.* | Text is in bold font. |
| **.BI** *t* | no | *t*=n.t.l. | Join words, alternating bold and italic. |
| **.BR** *t* | no | *t*=n.t.l. | Join words, alternating bold and roman. |
| **.DT** | no | .5i 1i... | Restore default tabs. |
| **.HP** *i* | yes | *i*=p.i.* | Begin paragraph with hanging indent. Set prevailing indent to *i*. |
| **.I** *t* | no | *t*=n.t.l. | Text is italic. |
| **.IB** *t* | no | *t*=n.t.l. | Join words, alternating italic and bold. |
| **.IP** *x i* | yes | *x*="" | Same as **.TP** with tag *x*. |
| **.IR** *t* | no | *t*=n.t.l. | Join words, alternating italic and roman. |
| **.LP** | yes | - | Begin left-aligned paragraph. Set prevailing indent to .5i. |
| **.PD** *d* | no | *d*=.4v | Set vertical distance between paragraphs. |
| **.PP** | yes | - | Same as **.LP**. |
| **.RE** | yes | - | End of relative indent. Restores prevailing indent. |
| **.RB** *t* | no | *t*=n.t.l. | Join words, alternating roman and bold. |
| **.RI** *t* | no | *t*=n.t.l. | Join words, alternating roman and italic. |
| **.RS** *i* | yes | *i*=p.i. | Start relative indent, increase indent by *i*. Sets prevailing indent to .5i for nested indents. |
| **.SB** *t* | no | - | Reduce size of text by 1 point, make text boldface. |
| **.SH** *t* | yes | *t*=n.t.l. | Section Heading. |
| **.SM** *t* | no | *t*=n.t.l. | Reduce size of text by 1 point. |
| **.SS** *t* | yes | *t*=n.t.l. | Section Subheading. |
| **.TH** *n s d f m* | yes | - | Begin reference page *n*, of of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets prevailing indent and tabs to .5i. |
| **.TP** *i* | yes | *i*=p.i. | Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*. |
| | * n.t.l. = next text line; p.i. = prevailing indent | | |
| **.TX** *t p* | no | - | Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*. |

**Conventions**

A typical manual page for a SunOS command or function is laid out as follows:

**.TH TITLE** [1-8]

The name of the command or function in upper-case, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

**.SH NAME** name (or comma-separated list of names) – one-line summary

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no **troff**(1) commands or escapes, and no macro requests. It is used to generate the **whatis**(1) database.

**.SH SYNOPSIS**

**Commands:**

The syntax of the command and its arguments, as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with an argument that you supply. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in roman face:

[ ] An argument, when surrounded by brackets is optional.

| Arguments separated by a vertical bar are exclusive. You can supply only item from such a list.

... Arguments followed by an elipsis can be repeated. When an elipsis follows a bracketed set, the expression within the brackets can be repeated.

**Functions:**

If required, the data declaration, or **#include** directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

**.SH DESCRIPTION**

A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output and standard error. Internals and implementation details are normally omitted. This section attempts to provide a succinct overview in answer to the question, "what does it do?"

Literal text from the synopsis appears in boldface, as do literal filenames and references to items that appear elsewhere in the *SunOS Reference Manual*. Arguments are italicized.

If a command interprets either subcommands or an input grammar, its command interface or input grammar is normally described in a USAGE section, which follows the OPTIONS section. (The DESCRIPTION section only describes the behavior of the command itself, not that of subcommands.)

**.SH OPTIONS**

The list of options along with a description of how each affects the command's operation.

**.SH FILES**

A list of files associated with the command or function.

**.SH SEE ALSO**

A comma-separated list of related manual pages, followed by references to other published materials.

**.SH DIAGNOSTICS**

A list of diagnostic messages and an explanation of each.

**.SH BUGS**

A description of limitations, known defects, and possible problems associated with the command or function.

**FILES**

**/usr/share/lib/tmac/tmac.an**

**SEE ALSO**

**man**(1), **nroff**(1), **troff**(1), **whatis**(1)

*Formatting Documents*.

**NAME**

me – macros for formatting papers

**SYNOPSIS**

**nroff –me** [ options ] file ...
**troff –me** [ options ] file ...

**DESCRIPTION**

This package of **nroff** and **troff** macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many **nroff** and **troff** requests are unsafe in conjunction with this package, however, these requests may be used with impunity after the first .pp:

.bp begin new page
.br break output line here
.sp n insert n spacing lines
.ls n (line spacing) n=1 single, n=2 double space
.na no alignment of right margin
.ce n center next n lines
.ul n underline next n lines
.sz +n add n to point size

Output of the **eqn, meqn, mefer,** and **tbl**(1) preprocessors for equations and tables is acceptable as input.

**REQUESTS**

In the following list, "initialization" refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete.

| Request | Initial Value | Cause Break | Explanation |
|---|---|---|---|
| .(c | - | yes | Begin centered block |
| .(d | - | no | Begin delayed text |
| .(f | - | no | Begin footnote |
| .(l | - | yes | Begin list |
| .(q | - | yes | Begin major quote |
| .(x*x* | - | no | Begin indexed item in index *x* |
| .(z | - | no | Begin floating keep |
| .)c | - | yes | End centered block |
| .)d | - | yes | End delayed text |
| .)f | - | yes | End footnote |
| .)l | - | yes | End list |
| .)q | - | yes | End major quote |
| .)x | - | yes | End index item |
| .)z | - | yes | End floating keep |
| .++ *m H* | - | no | Define paper section. *m defines the part of the paper, and can be* **C** (chapter), **A** (appendix), **P** (preliminary, for instance, abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one). |
| .+c *T* | - | yes | Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title. |
| .1c | 1 | yes | One column format on a new page. |
| .2c | 1 | yes | Two column format. |
| .EN | - | yes | Space after equation produced by **eqn** or **meqn**. |
| .EQ *x y* | - | yes | Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation. |
| .GE | - | yes | End *gremlin* picture. |
| .GS | - | yes | Begin *gremlin* picture. |

| | | | |
|---|---|---|---|
| .PE | - | yes | End *pic* picture. |
| .PS | - | yes | Begin *pic* picture. |
| .TE | - | yes | End table. |
| .TH | - | yes | End heading section of table. |
| .TS *x* | - | yes | Begin table; if *x* is *H* table has repeated heading. |
| .ac *A N* | - | no | Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization. |
| .b *x* | no | no | Print *x* in boldface; if no argument switch to boldface. |
| .ba +*n* | 0 | yes | Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs). |
| .bc | no | yes | Begin new column |
| .bi *x* | no | no | Print *x* in bold italics (nofill only) |
| .bu | - | yes | Begin bulleted paragraph |
| .bx *x* | no | no | Print *x* in a box (nofill only). |
| .ef *'x'y'z* | '''' | no | Set even footer to x y z |
| .eh *'x'y'z* | '''' | no | Set even header to x y z |
| .fo *'x'y'z* | '''' | no | Set footer to x y z |
| .hx | - | no | Suppress headers and footers on next page. |
| .he *'x'y'z* | '''' | no | Set header to x y z |
| .hl | - | yes | Draw a horizontal line |
| .i *x* | no | no | Italicize *x;* if *x* missing, italic text follows. |
| .ip *x y* | no | yes | Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5). |
| .lp | yes | yes | Start left-blocked paragraph. |
| .lo | - | no | Read in a file of local macros of the form .*x*. Must be given before initialization. |
| .np | 1 | yes | Start numbered paragraph. |
| .of *'x'y'z* | '''' | no | Set odd footer to x y z |
| .oh *'x'y'z* | '''' | no | Set odd header to x y z |
| .pd | - | yes | Print delayed text. |
| .pp | no | yes | Begin paragraph. First line indented. |
| .r | yes | no | Roman text follows. |
| .re | - | no | Reset tabs to default values. |
| .sc | no | no | Read in a file of special characters and diacritical marks. Must be given before initialization. |
| .sh *n x* | - | yes | Section head follows, font automatically bold. *n* is level of section, *x* is title of section. |
| .sk | no | no | Leave the next page blank. Only one page is remembered ahead. |
| .sm *x* - | *no* | | *Set x* in a smaller pointsize. |
| .sz +*n* | 10p | no | Augment the point size by *n* points. |
| .th | no | no | Produce the paper in thesis format. Must be given before initialization. |
| .tp | no | yes | Begin title page. |
| .u *x* | - | no | Underline argument (even in **troff**). (Nofill only). |
| .uh | - | yes | Like .sh but unnumbered. |
| .xp *x* | - | no | Print index *x*. |

**FILES**

**/usr/share/lib/tmac/tmac.e**
**/usr/share/lib/me/***

**SEE ALSO**

**eqn**(1), **nroff**(1), **troff**(1), **refer**(1), **tbl**(1)

*Formatting Documents*

## NAME

ms – text formatting macros

## SYNOPSIS

**nroff** –**ms** [ *options* ] *filename* ...

**troff** –**ms** [ *options* ] *filename* ...

## DESCRIPTION

This package of **nroff**(1) and **troff**(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through **col**(1V). All external –**ms** macros are defined below.

Note: this –**ms** macro package is an extended version written at Berkeley and is a superset of the standard –**ms** macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many **nroff** and **troff** requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

| | |
|---|---|
| **.bp** | begin new page |
| **.br** | break output line |
| **.sp** *n* | insert n spacing lines |
| **.ce** *n* | center next n lines |
| **.ls** *n* | line spacing: *n*=1 single, *n*=2 double space |
| **.na** | no alignment of right margin |

Font and point size changes with **\f** and **\s** are also allowed; for example, **\fIword\fR** will italicize *word*. Output of the **tbl**(1), **eqn**(1) and **refer**(1) preprocessors for equations, tables, and references is acceptable as input.

## REQUESTS

| Macro Name | Initial Value | Break? Reset? | Explanation |
|---|---|---|---|
| **.AB** *x* | – | y | begin abstract; if *x*=no do not label abstract |
| **.AE** | – | y | end abstract |
| **.AI** | – | y | author's institution |
| **.AM** | – | n | better accent mark definitions |
| **.AU** | – | y | author's name |
| **.B** *x* | – | n | embolden *x*; if no *x*, switch to boldface |
| **.B1** | – | y | begin text to be enclosed in a box |
| **.B2** | – | y | end boxed text and print it |
| **.BT** | date | n | bottom title, printed at foot of page |
| **.BX** *x* | – | n | print word *x* in a box |
| **.CM** | if t | n | cut mark between pages |
| **.CT** | – | y,y | chapter title: page number moved to CF (TM only) |
| **.DA** *x* | if n | n | force date *x* at bottom of page; today if no *x* |
| **.DE** | – | y | end display (unfilled text) of any kind |
| **.DS** *x y* | I | y | begin display with keep; *x*=I, L, C, B; *y*=indent |
| **.ID** *y* | 8n,.5i | y | indented display with no keep; *y*=indent |
| **.LD** | – | y | left display with no keep |
| **.CD** | – | y | centered display with no keep |
| **.BD** | – | y | block display; center entire block |
| **.EF** *x* | – | n | even page footer *x* (3 part as for **.tl**) |
| **.EH** *x* | – | n | even page header *x* (3 part as for **.tl**) |
| **.EN** | – | y | end displayed equation produced by **eqn** |

| | | | |
|---|---|---|---|
| **.EQ** *x y* | – | y | break out equation; *x*=L,I,C; *y*=equation number |
| **.FE** | – | n | end footnote to be placed at bottom of page |
| **.FP** | – | n | numbered footnote paragraph; may be redefined |
| **.FS** *x* | – | n | start footnote; *x* is optional footnote label |
| **.HD** | undef | n | optional page header below header margin |
| **.I** *x* | – | n | italicize *x*; if no *x*, switch to italics |
| **.IP** *x y* | – | y,y | indented paragraph, with hanging tag *x*; *y*=indent |
| **.IX** *x y* | – | y | index words *x y* and so on (up to 5 levels) |
| **.KE** | – | n | end keep of any kind |
| **.KF** | – | n | begin floating keep; text fills remainder of page |
| **.KS** | – | y | begin keep; unit kept together on a single page |
| **.LG** | – | n | larger; increase point size by 2 |
| **.LP** | – | y,y | left (block) paragraph. |
| **.MC** *x* | – | y,y | multiple columns; *x*=column width |
| **.ND** *x* | if t | n | no date in page footer; *x* is date on cover |
| **.NH** *x y* | – | y,y | numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y* |
| **.NL** | 10p | n | set point size back to normal |
| **.OF** *x* | – | n | odd page footer *x* (3 part as for **.tl**) |
| **.OH** *x* | – | n | odd page header *x* (3 part as for **.tl**) |
| **.P1** | if TM | n | print header on first page |
| **.PP** | – | y,y | paragraph with first line indented |
| **.PT** | - - | n | page title, printed at head of page |
| **.PX** *x* | – | y | print index (table of contents); *x*=no suppresses title |
| **.QP** | – | y,y | quote paragraph (indented and shorter) |
| **.R** | on | n | return to Roman font |
| **.RE** | 5n | y,y | retreat: end level of relative indentation |
| **.RP** *x* | – | n | released paper format; *x*=no stops title on first page |
| **.RS** | 5n | y,y | right shift: start level of relative indentation |
| **.SH** | – | y,y | section header, in boldface |
| **.SM** | – | n | smaller; decrease point size by 2 |
| **.TA** | 8n,5n | n | set TAB characters to 8n 16n ... (**nroff**) 5n 10n ... (**troff**) |
| **.TC** *x* | – | y | print table of contents at end; *x*=no suppresses title |
| **.TE** | – | y | end of table processed by **tbl** |
| **.TH** | – | y | end multi-page header of table |
| **.TL** | – | y | title in boldface and two points larger |
| **.TM** | off | n | UC Berkeley thesis mode |
| **.TS** *x* | – | y,y | begin table; if *x*=H table has multi-page header |
| **.UL** *x* | – | n | underline *x*, even in **troff** |
| **.UX** *x* | – | n | UNIX; trademark message first time; *x* appended |
| **.XA** *x y* | – | y | another index entry; *x*=page or no for none; *y*=indent |
| **.XE** | – | y | end index entry (or series of **.IX** entries) |
| **.XP** | – | y,y | paragraph with first line exdented, others indented |
| **.XS** *x y* | – | y | begin index entry; *x*=page or no for none; *y*=indent |
| **.1C** | on | y,y | one column format, on a new page |
| **.2C** | – | y,y | begin two column format |
| **.]–** | – | n | beginning of **refer** reference |
| **.[ 0** | – | n | end of unclassifiable type of reference |
| **.[ N** | – | n | N= 1:journal-article, 2:book, 3:book-article, 4:report |

REGISTERS

Formatting distances can be controlled in **–ms** by means of built-in number registers. For example, this sets the line length to 6.5 inches:

**.nr LL 6.5i**

Here is a table of number registers and their default values:

| Name | Register Controls | Takes Effect | Default |
|---|---|---|---|
| **PS** | point size | paragraph | 10 |
| **VS** | vertical spacing | paragraph | 12 |
| **LL** | line length | paragraph | 6i |
| **LT** | title length | next page | same as **LL** |
| **FL** | footnote length | next **.FS** | 5.5i |
| **PD** | paragraph distance | paragraph | 1v (if n), .3v (if t) |
| **DD** | display distance | displays | 1v (if n), .5v (if t) |
| **PI** | paragraph indent | paragraph | 5n |
| **QI** | quote indent | next **.QP** | 5n |
| **FI** | footnote indent | next **.FS** | 2n |
| **PO** | page offset | next page | 0 (if n), ~1i (if t) |
| **HM** | header margin | next page | 1i |
| **FM** | footer margin | next page | 1i |
| **FF** | footnote format | next **.FS** | 0 (1, 2, 3 available) |

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting **FF** to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an **.IP**-like footnote paragraph.

Here is a list of string registers available in **–ms**; they may be used anywhere in the text:

| Name | String's Function |
|---|---|
| ***Q** | quote (" in **nroff**, " in **troff**) |
| ***U** | unquote (" in **nroff**, " in **troff**) |
| ***–** | dash (-- in **nroff**, — in **troff**) |
| ***(MO** | month (month of the year) |
| ***(DY** | day (current date) |
| ****** | automatically numbered footnote |
| ***´** | acute accent (before letter) |
| ***`** | grave accent (before letter) |
| ***^** | circumflex (before letter) |
| ***,** | cedilla (before letter) |
| ***:** | umlaut (before letter) |
| ***~** | tilde (before letter) |

When using the extended accent mark definitions available with **.AM**, these strings should come after, rather than before, the letter to be accented.

## FILES

**/usr/share/lib/tmac/tmac.s**
**/usr/share/lib/ms/ms.???**

## SEE ALSO

**col**(1V), **eqn**(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1)

*Formatting Documents*

## BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

## NAME

intro – introduction to system maintenance and operation commands

## DESCRIPTION

This section contains information related to system bootstrapping, operation and maintenance. It describes all the server processes and daemons that run on the system, as well as standalone (PROM monitor) programs.

Disk formatting and labeling is done by **format**(8S). Bootstrapping of the system is described in **boot**(8S) and **init**(8). The standard set of commands run by the system when it boots is described in **rc**(8). Related commands include those that check the consistency of file systems, **fsck**(8); those that mount and unmount file systems, **mount**(8); add swap devices, **swapon**(8); force completion of outstanding file system I/O, **sync**(2); shutdown or reboot a running system **shutdown**(8), **halt**(8), and **reboot**(8); and, set the time on a machine from the time on another machine **rdate**(8).

Creation of file systems is discussed in **mkfs**(8) and **newfs**(8). File system performance parameters can be adjusted with **tunefs**(8). File system backups and restores are described in **dump**(8) and **restore**(8).

Procedures for adding new users to a system are described in **adduser**(8), using **vipw**(8) to lock the password file during editing. **crash**(8S) which describes what happens when the system crashes, **savecore**(8) and **analyze**(8), which can be used to analyze system crash dumps. Occasionally useful as adjuncts to the **fsck**(8) file system repair program are **clri**(8), **dcheck**(8), **icheck**(8), and **ncheck**(8).

Configuring a new version of the kernel requires using the program **config**(8); major system bootstraps often require the use of **mkproto**(8). New devices are added to the **/dev** directory (once device drivers are configured into the system) using **makedev**(8) and **mknod**(8). The **installboot**(8S) command can be used to install freshly compiled programs. The **catman**(8) command preformats the on-line manual pages.

Resource accounting is enabled by the **accton** command, and summarized by **sa**(8). Login time accounting is performed by **ac**(8). Disk quotas are managed using **quot**(8), **quotacheck**(8), **quotaon**(8), and **repquota**(8).

A number of servers and daemon processes are described in this section. The **update**(8) daemon forces delayed file system I/O to occur and **cron**(8) runs periodic events (such as removing temporary files from the disk periodically). The **syslogd**(8) daemon maintains the system error log. The **init**(8) process is the initial process created when the system boots. It manages the reboot process and creates the initial login prompts on the various system terminals, using **getty**(8). The Internet super-server **inetd**(8C) invokes all other internet servers as needed. These servers include the remote shell servers **rshd**(8C) and **rexecd**(8C), the remote login server **rlogind**(8C), the FTP and TELNET daemons **ftpd**(8C), and **telnetd**(8C), the TFTP daemon **tftpd**(8C), and the mail arrival notification daemon **comsat**(8C). Other network daemons include the 'load average/who is logged in' daemon **rwhod**(8C), the routing daemon **routed**(8C), and the mail daemon **sendmail**(8).

If network protocols are being debugged, then the protocol debugging trace program **trpt**(8C) is often useful. Remote magnetic tape access is provided by **rsh** and **rmt**(8C). Remote line printer access is provided by **lpd**(8), and control over the various print queues is provided by **lpc**(8). Printer cost-accounting is done through **pac**(8).

Network host tables may be gotten from the ARPA NIC using **gettable**(8C) and converted to UNIX-system-usable format using **htable**(8).

### RPC and NFS daemons

RPC and NFS daemons include:

| | |
|---|---|
| **portmap** | used by RPC based services. |
| **ypbind** | used by the Yellow Pages to locate the Yellow Pages server. |
| **biod** | used by NFS clients to read ahead to, and write behind from, network file systems. |
| **nfsd** | the NFS server process that responds to NFS requests on NFS server machines. |
| **ypserv** | the Yellow Pages server, typically run on each NFS server. |
| **rstatd** | the server counterpart of the remote speedometer tools. |

**mountd** the **mount** server that runs on NFS server machines and responds to requests by other machines to mount file systems.

**rwalld** used for broadcasting messages over the network.

LIST OF MAINTENANCE COMMANDS

| Name | Appears on Page | Description |
|---|---|---|
| **ac** | **ac**(8) | login accounting |
| **accton** | **sa**(8) | system accounting |
| **adbgen** | **adbgen**(8) | generate adb script |
| **adduser** | **adduser**(8) | procedure for adding new users |
| **arp** | **arp**(8C) | address resolution display and control |
| **audit** | **audit**(8) | audit trail maintenance |
| **audit_warn** | **audit_warn**(8) | audit space low warning script |
| **auditd** | **auditd**(8) | audit daemon |
| **automount** | **automount**(8) | automatically mount NFS file systems |
| **biod** | **nfsd**(8) | NFS daemons |
| **boot** | **boot**(8S) | start the system kernel or a standalone program |
| **bootparamd** | **bootparamd**(8) | boot parameter server |
| **captoinfo** | **captoinfo**(8V) | convert a termcap description into a terminfo descriptio |
| **catman** | **catman**(8) | create the cat files for the manual |
| **chown** | **chown**(8) | change owner |
| **chroot** | **chroot**(8) | change root directory for a command |
| **client** | **client**(8) | add/remove diskless systems |
| **clri** | **clri**(8) | clear i-node |
| **comsat** | **comsat**(8C) | biff server |
| **config** | **config**(8) | build system configuration files |
| **crash** | **crash**(8S) | what happens when the system crashes |
| **cron** | **cron**(8) | clock daemon |
| **dcheck** | **dcheck**(8) | file system directory consistency check |
| **dkinfo** | **dkinfo**(8) | report information about a disk's geometry and partitior |
| **dmesg** | **dmesg**(8) | collect system diagnostic messages to form error log |
| **dump** | **dump**(8) | incremental file system dump |
| **dumpfs** | **dumpfs**(8) | dump file system information |
| **edquota** | **edquota**(8) | edit user quotas |
| **eeprom** | **eeprom**(8S) | Sun-3 EEPROM display and load utility |
| **etherd** | **etherd**(8C) | Ethernet statistics server |
| **etherfind** | **etherfind**(8C) | find packets on Ethernet |
| **exportfs** | **exportfs**(8) | export and unexport directories to NFS clients |
| **fastboot** | **fastboot**(8) | reboot/halt the system without checking the disks |
| **fasthalt** | **fastboot**(8) | reboot/halt the system without checking the disks |
| **fingerd** | **fingerd**(8C) | remote user information server |
| **format** | **format**(8S) | disk partitioning and maintenance utility |
| **fparel** | **fparel**(8) | Sun FPA online reliability tests |
| **fpaversion** | **fpaversion**(8) | print FPA version |
| **fsck** | **fsck**(8) | file system consistency check and interactive repair |
| **fsirand** | **fsirand**(8) | install random inode generation numbers |
| **ftpd** | **ftpd**(8C) | DARPA Internet File Transfer Protocol server |
| **gettable** | **gettable**(8C) | get DoD Internet format host table from a host |
| **getty** | **getty**(8) | set terminal mode |
| **gpconfig** | **gpconfig**(8) | initialize the Graphics Processor |
| **grpck** | **grpck**(8) | check group database entries |
| **halt** | **halt**(8) | stop the processor |
| **htable** | **htable**(8) | convert DoD Internet format host table |

| | | |
|---|---|---|
| **icheck** | **icheck**(8) | file system storage consistency check |
| **ifconfig** | **ifconfig**(8C) | configure network interface parameters |
| **inetd** | **inetd**(8C) | Internet services daemon |
| **infocmp** | **infocmp**(8V) | compare or print out terminfo descriptions |
| **init** | **init**(8) | process control initialization |
| **installboot** | **boot**(8S) | start the system kernel or a standalone program |
| **iostat** | **iostat**(8) | report I/O statistics |
| **ipallocd** | **ipallocd**(8C) | Ethernet-to-IP address allocator |
| **kadb** | **kadb**(8S) | adb-like kernel and standalone-program debugger |
| **keyenvoy** | **keyenvoy**(8C) | talk to keyserver |
| **keyserv** | **keyserv**(8C) | server for storing public and private keys |
| **kgmon** | **kgmon**(8) | generate a dump of the operating system's profile buffers |
| **ldconfig** | **ldconfig**(8) | configure cache for **ld.so** |
| **link** | **link**(8) | exercise link and unlink system calls |
| **lockd** | **lockd**(8C) | network lock daemon |
| **logintool** | **logintool**(8) | graphic login interface |
| **lpc** | **lpc**(8) | line printer control program |
| **lpd** | **lpd**(8) | printer daemon |
| **mailstats** | **mailstats**(8) | print statistics collected by sendmail |
| **MAKEDBM** | **makedbm**(8) | make a Yellow Pages dbm file |
| **MAKEDEV** | **makedev**(8) | make system special files |
| **makekey** | **makekey**(8) | generate encryption key |
| **mc68881version** | **mc68881version**(8) | print the MC68881 mask number and approximate clock rate |
| **mconnect** | **mconnect**(8) | connect to SMTP mail server socket |
| **mkfs** | **mkfs**(8) | construct a file system |
| **mknod** | **mknod**(8) | build special file |
| **mkproto** | **mkproto**(8) | construct a prototype file system |
| **modload** | **modload**(8) | load a loadable module |
| **modstat** | **modstat**(8) | display status of loadable modules |
| **modunload** | **modunload**(8) | unload a loadable module |
| **monitor** | **monitor**(8S) | system ROM monitor |
| **mount** | **mount**(8) | mount and dismount filesystems |
| **mountd** | **mountd**(8C) | NFS mount request server |
| **named** | **named**(8C) | Internet domain name server |
| **ncheck** | **ncheck**(8) | generate names from i-numbers |
| **ndbootd** | **ndbootd**(8C) | ND boot block server |
| **netconfig** | **netconfig**(8C) | PNP boot service |
| **netstat** | **netstat**(8C) | show network status |
| **newaliases** | **newaliases**(8) | rebuild the data base for the mail aliases file |
| **newfs** | **newfs**(8) | construct a new file system |
| **newkey** | **newkey**(8) | create a new key in the publickey database |
| **nfsd** | **nfsd**(8) | NFS daemons |
| **nfsstat** | **nfsstat**(8C) | Network File System statistics |
| **pac** | **pac**(8) | printer/plotter accounting information |
| **ping** | **ping**(8C) | send ICMP ECHO_REQUEST packets to network hosts |
| **pnp.s386** | **pnpboot**(8C) | PNP diskless boot service |
| **pnpboot** | **pnpboot**(8C) | PNP diskless boot service |
| **pnpd** | **pnpd**(8C) | PNP daemon |
| **portmap** | **portmap**(8C) | DARPA port to RPC program number mapper |
| **praudit** | **praudit**(8) | print contents of an audit trail file |
| **pstat** | **pstat**(8) | print system facts |
| **pwck** | **pwck**(8) | check password database entries |
| **pwdauthd** | **pwdauthd**(8C) | server for authenticating passwords |

| | | |
|---|---|---|
| **quot** | **quot**(8) | summarize file system ownership |
| **quotacheck** | **quotacheck**(8) | file system quota consistency checker |
| **quotaoff** | **quotaon**(8) | turn file system quotas on and off |
| **quotaon** | **quotaon**(8) | turn file system quotas on and off |
| **rarpd** | **rarpd**(8C) | DARPA Reverse Address Resolution Protocol service |
| **rc.boot** | **rc**(8) | command scripts for auto-reboot and daemons |
| **rc.local** | **rc**(8) | command scripts for auto-reboot and daemons |
| **rc** | **rc**(8) | command scripts for auto-reboot and daemons |
| **rdate** | **rdate**(8C) | set system date from a remote host |
| **rdump** | **dump**(8) | incremental file system dump |
| **reboot** | **reboot**(8) | restart the operating system |
| **renice** | **renice**(8) | alter priority of running processes |
| **repquota** | **repquota**(8) | summarize quotas for a file system |
| **restore** | **restore**(8) | incremental file system restore |
| **rexd** | **rexd**(8C) | RPC-based remote execution server |
| **rexecd** | **rexecd**(8C) | remote execution server |
| **rlogind** | **rlogind**(8C) | remote login server |
| **rmail** | **rmail**(8C) | handle remote mail received via uucp |
| **rmt** | **rmt**(8C) | remote magtape protocol module |
| **route** | **route**(8C) | manually manipulate the routing tables |
| **routed** | **routed**(8C) | network routing daemon |
| **rpcinfo** | **rpcinfo**(8C) | report RPC information |
| **rquotad** | **rquotad**(8C) | remote quota server |
| **rrestore** | **restore**(8) | incremental file system restore |
| **rshd** | **rshd**(8C) | remote shell server |
| **rstatd** | **rstatd**(8C) | kernel statistics server |
| **rusersd** | **rusersd**(8C) | network username server |
| **rwalld** | **rwalld**(8C) | network rwall server |
| **rwhod** | **rwhod**(8C) | system status server |
| **sa** | **sa**(8) | system accounting |
| **savecore** | **savecore**(8) | save a core dump of the operating system |
| **sendmail** | **sendmail**(8) | send mail over the internet |
| **setup_client** | **setup_client**(8) | create or remove a nfs client on a 4.0 server. |
| **setup_exec** | **setup_exec**(8) | install architecture-dependent executable files |
| **showmount** | **showmount**(8) | show all remote mounts |
| **shutdown** | **shutdown**(8) | close down the system at a given time |
| **spray** | **spray**(8C) | spray packets |
| **sprayd** | **sprayd**(8C) | spray server |
| **statd** | **statd**(8C) | network status monitor |
| **sticky** | **sticky**(8) | persistent text and append-only directories |
| **suninstall** | **suninstall**(8) | SunOS software installation program |
| **swapon** | **swapon**(8) | specify additional device for paging and swapping |
| **sysdiag** | **sysdiag**(8) | system diagnostics |
| **syslogd** | **syslogd**(8) | log system messages |
| **talkd** | **talkd**(8C) | server for talk program |
| **telnetd** | **telnetd**(8C) | DARPA TELNET protocol server |
| **tftpd** | **tftpd**(8C) | DARPA Trivial File Transfer Protocol server |
| **tic** | **tic**(8V) | terminfo compiler |
| **timed** | **timed**(8C) | DARPA Time server |
| **tnamed** | **tnamed**(8C) | DARPA Trivial name server |
| **trpt** | **trpt**(8C) | transliterate protocol trace |
| **tunefs** | **tunefs**(8) | tune up an existing file system |
| **umount** | **mount**(8) | mount and dismount filesystems |

| | | |
|---|---|---|
| **unconfigure** | **unconfigure**(8) | reset the network configuration for a system |
| **unlink** | **link**(8) | exercise link and unlink system calls |
| **update** | **update**(8) | periodically update the super block |
| **uuclean** | **uuclean**(8C) | uucp spool directory clean-up |
| **vipw** | **vipw**(8) | edit the password file |
| **vmstat** | **vmstat**(8) | report virtual memory statistics |
| **ypbind** | **ypserv**(8) | Yellow Pages server and binder processes |
| **ypinit** | **ypinit**(8) | build and install Yellow Pages database |
| **ypmake** | **ypmake**(8) | rebuild Yellow Pages database |
| **yppasswdd** | **yppasswdd**(8C) | server for modifying Yellow Pages password file |
| **yppoll** | **yppoll**(8) | what version of a YP map is at a YP server host |
| **yppush** | **yppush**(8) | force propagation of a changed YP map |
| **ypserv** | **ypserv**(8) | Yellow Pages server and binder processes |
| **ypset** | **ypset**(8) | point ypbind at a particular server |
| **ypupdated** | **ypupdated**(8C) | server for changing yp information |
| **ypwhich** | **ypwhich**(8) | what machine is the YP server? |
| **ypxfr** | **ypxfr**(8) | transfer YP map from a YP server to here |
| **zdump** | **zdump**(8) | time zone dumper |
| **zic** | **zic**(8) | time zone compiler |

## NAME

ac – login accounting

## SYNOPSIS

**/usr/etc/ac** [ **–w** *wtmp* ] [ **–p** ] [ **–d** ] [ *people* ] ...

## DESCRIPTION

**ac** produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced.

The accounting file **/var/adm/wtmp** is maintained by **init**(8) and **login**(1). Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

## OPTIONS

**–w** Specify an alternate *wtmp* file.

**–p** Print individual totals; without this option, only totals are printed.

**–d** Printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, **/var/adm/wtmp** is used.

## FILES

**/var/adm/wtmp**

## SEE ALSO

**login**(1), **utmp**(5), **init**(8), **sa**(8)

NAME

adbgen – generate adb script

SYNOPSIS

**/usr/lib/adb/adbgen** *filename*.**adb** ...

DESCRIPTION

**adbgen** makes it possible to write **adb**(1) scripts that do not contain hard-coded dependencies on structure member offsets. The input to **adbgen** is a file named *filename*.**adb** which contains **adbgen** header information, then a null line, then the name of a structure, and finally an **adb** script. **adbgen** only deals with one structure per file; all member names are assumed to be in this structure. The output of **adbgen** is an **adb** script in *filename*. **adbgen** operates by generating a C program which determines structure member offsets and sizes, which in turn generates the **adb** script.

The header lines, up to the null line, are copied verbatim into the generated C program. Typically these include C **#include** statements to include the header files containing the relevant structure declarations.

The **adb** script part may contain any valid **adb** commands (see **adb**(1)), and may also contain **adbgen** requests, each enclosed in {}s. Request types are:

1 Print a structure member. The request form is {*member*,*format*}. *member* is a member name of the *structure* given earlier, and *format* is any valid **adb** format request. For example, to print the **p_pid** field of the *proc* structure as a decimal number, you would write **{p_pid,d}**.

2 Reference a structure member. The request form is {*****member,*base*}. *member* is the member name whose value is desired, and *base* is an **adb** register name which contains the base address of the structure. For example, to get the **p_pid** field of the *proc* structure, you would get the proc structure address in an **adb** register, say <f, and write **{*p_pid,<f}**.

3 Tell **adbgen** that the offset is ok. The request form is {OFFSETOK}. This is useful after invoking another **adb** script which moves the **adb** *dot*.

4 Get the size of the *structure*. The request form is {SIZEOF}. **adbgen** replaces this request with the size of the structure. This is useful in incrementing a pointer to step through an array of structures.

5 Get the offset to the end of the structure. The request form is {END}. This is useful at the end of the structure to get **adb** to align the **dot** for printing the next structure member.

**adbgen** keeps track of the movement of the **adb** *dot* and emits **adb** code to move forward or backward as necessary before printing any structure member in a script. **adbgen**'s model of the behavior of **adb**'s **dot** is simple: it is assumed that the first line of the script is of the form *struct_address/adb text* and that subsequent lines are of the form *+/adb text*. This causes the *adb* **dot** to move in a sane fashion. **adbgen** does not check the script to ensure that these limitations are met. **adbgen** also checks the size of the structure member against the size of the **adb** format code and warns you if they are not equal.

EXAMPLE

If there were an include file **x.h** which contained:

```
struct x {
        char    *x_cp;
        char    x_c;
        int     x_i;
};
```

Then an **adbgen** file (call it **script.adb**) to print it would be:

```
#include "x.h"
x
./"x_cp"16t"x_c"8t"x_i"n{x_cp,X}{x_c,C}{x_i,D}
```

After running **adbgen** the output file **script** would contain:

```
16t"x_c"8t"x_i"nXC+D"" ./"x_cp"16t"x_c"8t"x_i"nXC+D
```

To invoke the script you would type:

```
x$<script
```

FILES

| | |
|---|---|
| **/usr/lib/adb/*** | **adb** scripts for debugging the kernel |

SEE ALSO

**adb**(1), **kadb**(8S)

*Debugging Tools*

BUGS

**adb** syntax is ugly; there should be a higher level interface for generating scripts.

Structure members which are bit fields cannot be handled because C will not give the address of a bit field. The address is needed to determine the offset.

DIAGNOSTICS

Warnings about structure member sizes not equal to **adb** format items and complaints about badly formatted requests. The C compiler complains if you reference a structure member that does not exist. It also complains about & before array names; these complaints may be ignored.

## NAME

adduser – procedure for adding new users

## DESCRIPTION

To add an account for a new user, the system administrator (or super-user):

- Create an entry for the new user in the system password files.
- Create a home directory for the user, and change ownership so the new user owns that directory.
- Optionally set up skeletal dot files for the new user (**.cshrc, .login, .profile...**).
- If the account is on a system running the YP name service, take additional measures.

## USAGE

### Making an Entry in the Password File

To add an entry for the new login name on a local host, first edit the **/etc/passwd** file — inserting a line for the new user. This must be done with the password file locked, for instance, by using **vipw**(8), and the insertion must be made above the line containing the string:

**+::0:0:::**

This line is used to indicate that additional accounts can be found in the YP.

To add an entry for the new login name on to the YP, add an identical line to the file **/etc/passwd** on the YP master server, and run **make**(1) in the directory **/var/yp** (see **ypmake**(8) for details) to propagate the change.

The new user is assigned a group and user ID number. User ID numbers (or userids, or UIDs) should be unique for each user and consistent across the NFS domain, since they control access to files. Group ID numbers (or groupids, or GIDs) need not be unique. Typically, users working on similar projects will assigned to the same group. The system staff is group 10 for historical reasons, and the super-user is in this group.

An entry for a new user "francine" would look like this:

**francine::235:20:& Featherstonehaugh:/usr/francine:/bin/csh**

Fields in each password-file entry are delimited by colons, and have the following meanings:

- Login name ("francine"). The login name is limited to eight characters in length.
- Encrypted password or the string *##name* if encrypted passwords are stored in the password adjunct file. Typically, if passwords are to be stored in the main password file, this field is left empty, so no password is needed when the user first logs in. If security demands a password, it should be assigned by running **passwd**(1) immediately after exiting the editor. The number of significant characters in a password is eight. (See **passwd**(1).)
- User ID. The UID is a number which identifies that user uniquely in the system. Files owned by the user have this number stored in their data blocks, and commands such as **ls**(1V) use it to look up the owner's login name. For this reason, you cannot randomly change this number. See **passwd**(5) for more information.
- Group ID. The UID number identifies the group to which the user belongs by default (although the user may belong to additional groups as well). All files that the user creates have this number stored in their data blocks, and commands such as **ls**(1V) use it to look up the group name. Group names and assignments are listed in the file **/etc/group** (which is described in **group**(5)) or in the YP group map.
- This field is called the GCOS field (from earlier implementation of the operating system) and is traditionally used to hold the user's full name. Some installations have other information encoded in this field. From this information we can tell that Francine's real name is 'Francine Featherstonehaugh'. The & in the entry is shorthand for the user's login name.

- User's home directory. This is the directory in which that user is "positioned" when they log in.
- Initial shell which this user will see on login. If this field is empty, **sh**(1) is used as the initial shell.

An entry for a new user "francine" would look like this:

**francine:::::lo:ad,+dw**

Fields in each password adjunct file entry are delimited by colons, and have the following meanings:

- Login name ("francine"). This name must match the login name in the password file.
- Encrypted password. Typically, this field is left empty when adding the line using the editor. **passwd**(1) should be run immediately after exiting the editor.
- The next three fields are the minimum label, the maximum label, and the default label. These fields should be left empty, since they are reserved for future use.
- The next two fields are for the always-audit flags and the never-audit flags. Always-audit flags specify which events guaranteed to be audited for that user. Never-audit flags specify which events are guaranteed not to be audited for that user. For a description of audit flags, see **audit_data**(5).

**Making a Home Directory**

As shown in the password file entry above, the name of Francine's home directory is to be **/usr/francine**. This directory must be created using **mkdir**(1), and Francine must be given ownership of it using **chown**(8), in order for her profile files to be read and executed, and to have control over access to it by other users:

**example# mkdir /usr/francine**
**example# /usr/etc/chown francine /usr/francine**

If running under NFS, the **mkdir**(1) and **chown**(8) commands must be performed on the NFS server.

**Setting Up Skeletal Profile Files**

New users often need assistance in setting up their profile files to initialize the terminal properly, configure their search path, and perform other desired functions at startup. Providing them with skeletal profile files saves time and interruptions for both the new user and the system administrator.

Such files as **.profile** (if they use **/usr/bin/sh** as the shell), or **.cshrc** and **.login** (if they use **/usr/bin/csh** as the shell), can include commands that are performed automatically at each login, or whenever a shell is invoked, such as **tset**(1). The ownership of these files must be changed to belong to the new user, either by running **su**(1) before making copies, or by using **chown**(8).

**FILES**

| | |
|---|---|
| **/etc/passwd** | password file |
| **/etc/group** | group file |
| **/etc/yp/src/passwd** | |
| **~/.cshrc** | |
| **~/.login** | |
| **~/.profile** | |

**SEE ALSO**

**csh**(1), **ls**(1v), **make**(1), **mkdir**(1), **passwd**(1), **sh**(1), **su**(1), **tset**(1), **audit**(2), **audit_control**(5), **audit_data**(5), **group**(5), **passwd**(5), **passwd.adjunct**(5), **audit**(8), **auditd**(8), **chown**(8), **vipw**(8), **ypmake**(8)

*Network Programming*

## NAME

arp – address resolution display and control

## SYNOPSIS

**arp** *hostname*

**arp** **–a** [ *vmunix* [ *kmem* ] ]

**arp** **–d** *hostname*

**arp** **–s** *hostname ether_address* [ **temp** ] [ **pub** ] [ **trail** ]

**arp** **–f** *filename*

## DESCRIPTION

The **arp** program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol (**arp**(4P)).

With no flags, the program displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation.

## OPTIONS

**–a** Display all of the current ARP entries by reading the table from the file *kmem* (default **/dev/kmem**) based on the kernel file *vmunix* (default **/vmunix**).

**–d** Delete an entry for the host called *hostname*. This option may only be used by the super-user.

**–s** Create an ARP entry for the host called *hostname* with the Ethernet address *ether_address*. The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless the word **temp** is given in the command. If the word **pub** is given, the entry will be published, for instance, this system will respond to ARP requests for *hostname* even though the hostname is not its own. The word **trail** indicates that trailer encapsulations may be sent to this host.

**–f** Read the file named *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form

*hostname ether_address* [ **temp** ] [ **pub** ] [ **trail** ]

with argument meanings as given above.

## SEE ALSO

**arp**(4P), **ifconfig**(8C)

NAME

audit – audit trail maintenance

SYNOPSIS

**audit** [ **–n**|**–s**|**–t** ]
**audit** **–d** *username*
**audit** **–u** *username audit_event_state*

AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

The **audit** command is the general administrator's interface to kernel auditing. The process audit state for a user can be temporarily or permanently altered. The audit daemon may be notified to read the contents of the **audit_control** file and re-initialize the current audit directory to the first directory listed in the **audit_control** file, or to open a new audit file in the current audit directory specified in the **audit_control** file as last read by the audit daemon. Auditing may also be terminated/disabled.

OPTIONS

**–n** Signal audit daemon to close the current audit file and open a new audit file in the current audit directory.

**–s** Signal audit daemon to read audit control file. The audit daemon stores the information internally.

**–t** Signal audit daemon to disable auditing and die.

**–d** *username*

Change the process audit state of all processes owned by *username*. This new process audit state is constructed from the system and user audit values as specified in the **audit_control** and **passwd.adjunct** files respectively.

**–u** *username audit_event_state*

Set the process audit state from *audit_event_state* for all current processes owned by *username*. See **audit_control**(5) for the format of the system audit value. The process audit state is one argument. Enclose the audit event state in quotes, or do not use SPACE characters in the process audit state specification. A new login session reconstructs the process audit state from the audit flags in the **audit_control** and **passwd.adjunct** files.

SEE ALSO

**audit**(2), **setuseraudit**(2), **getauditflags**(3), **getfauditflags**(3), **audit_control**(5), **passwd.adjunct**(5)

**NAME**

audit_warn – audit space low warning script

**SYNOPSIS**

**/etc/security/audit/audit_warn logfile**

**DESCRIPTION**

The *audit_warn* shell script is used to take appropriate action when audit filesystem space is running low. This script is run when the audit filesystem free disk space drops below the value *minfree* as specified in the file *audit_control*. The script is passed the name of the current audit log file.

The *audit_warn* script included with the installation tape issues the command "audit -n", which tells the audit daemon to switch to the next audit directory.

**SEE ALSO**

auditd(8), audit(8), audit.log(5), audit_control(5)

**NAME**

auditd – audit daemon

**SYNOPSIS**

**/etc/auditd** [ *username* ]

**DESCRIPTION**

The **auditd** program is the daemon process that drives audit log generation. **auditd** runs as root unless the *username* parameter specifies another valid user ID. Use of this parameter is recommended to insure that **auditd** works over NFS and that the audit data is secure.

After reading the **audit_control** file, **auditd** opens an audit log file in the first directory specified. If there is an error opening this file, the daemon tries successive directories until successful. Then the daemon invokes the **auditsvc**(2) system call to initiate audit record logging to the audit log file. The system call does not return until:

- disk space is low on the audit filesystem
- there is an error writing the audit log file or
- the daemon receives a signal

**auditd** simply ignores most signals and re-issues the **auditsvc**( ) system call. However, for SIGHUP, the daemon re-reads the **audit_control** file, closes the current audit log file, and opens a new audit log file based on the new directory list.

If the **auditsvc**( ) system call returns because of low disk space, **auditd** invokes the shell script **audit_warn**(8) with the name of the current audit log file, then returns to the **auditsvc**( ) system call to continue auditing using the same audit file.

When the **auditsvc**( ) system call returns because of an error, **auditd** recovers from the problem by closing the current audit log file and opening a new audit log file in the next directory in the list. This recovers from most errors, such as lack of disk space, or file server crashes.

Should the audit daemon run out of audit directories, it attempts to recover. It suspends itself for a few seconds, and then re-reads the **audit_control** file. It then tries again to open audit logs in the specified audit directory list.

**SEE ALSO**

**auditsvc**(2), **audit_control**(5), **audit.log**(5), **audit**(8), **audit_warn**(8)

## NAME

automount – automatically mount NFS file systems

## SYNOPSIS

**automount** [ **–mnT** ] [ **–tl** *duration* ] [ **–tm** *interval* ] [ **–tw** *interval* ]
[ *directory mapname* [ *–mount-options* ] ] ...

## DESCRIPTION

**automount** is a daemon that will automatically and transparently mount an NFS file system whenever a file or directory within in that system is opened. **automount** forks a daemon, which appears to be an NFS server to the kernel; lookups on the specified *directory* are intercepted by this daemon, which uses the map contained in *mapname* to determine a server, exported file system, and appropriate mount options for a given file system. The named map can either be a file on the local system, or a Yellow Pages map. *directory* is a full pathname starting with a '/'.

When supplied, *–mount-options* consists of the leading – and a comma-separate list of **mount**(8) options; if mount options are specified in the map, however, those in the map take precedence.

Once mounted, members of the *directory* are made available using a symbolic link to the real mount point within a temporary directory.

If *directory* does not exist, the daemon creates it, and then removes it automatically when the daemon exits.

Since the name-to-location binding is dynamic, updates to a Yellow Pages map are transparent to the user. This obviates the need to "pre-mount" shared file systems for applications that have "hard coded" references to files. It also obviates the need to maintain records of which hosts must be mounted for what applications.

### Maps

**automount** looks first for the indicated *mapname* in a file by that name. If there is no such file, it looks for a YP map by that name.

An automount map is composed of a list of mappings, with one mapping per line. Each mapping is composed of the following fields:

*basename* [*–mount-options*] *location* [...]

where *basename* is the name of a subdirectory within the *directory* specified in the **automount** command line (not a relative pathname). The *location* field consists of an entry of the form:

*host*:*directory*[:*subdir*]

where *host* is the name of the host from which to mount the file system, *directory* is the pathname of the directory to mount, and *subdir*, when supplied, is the name of a subdirectory to which the symbolic link is made. This can be used to prevent duplicate mounts in cases where multiple directories in the same remote file system are accessed.

The contents of a YP map can be included within a map by adding an entry of the form:

*+mapname*

A mapping can be continued across line breaks using a \ as the last character before the NEWLINE. Comments begin with a # and end at the subsequent NEWLINE.

If more than one *location* is supplied, there is no guarantee as to which location will be used; the first location to respond to the mount request gets mounted. The *mount-options* field can be used to supply options to the **mount**(8) command for the mounted file system.

**Special Maps**

There are two special maps currently available. The **–hosts** map uses the Yellow Pages **hosts.byname** map to locate a remote host when the hostname is specified as a subdirectory of *directory*. This map specifies mounts of all exported file systems from any host. For instance, if the following **automount** command is already in effect:

**automount /net –hosts**

then a reference to **/net/hermes/usr** would initiate an automatic mount of all file systems from **hermes** that **automount** can mount; references to a directory under **/net/hermes** will refer to the corresponding directory on **hermes**. The **–passwd** map uses the **passwd**(5) database to attempt to locate the home directory of a user. For instance, if the following **automount** command is already in effect:

**automount /homes –passwd**

then if the home directory shown in the passwd entry for the user *username* has the form */dir/server/username*, and **server** matches the host system on which that directory resides, references to files in **/homes/***username* result in the file system containing that directory being mounted if necessary, and all such references will refer to that user's home directory.

**Configuration**

**automount** normally consults the **auto.master** Yellow Pages configuration database for a list of initial *directory* to *mapname* pairs, and sets up automatic mounts for them in addition to those given on the command line; if there are duplications, the command-line arguments take precedence. (Note that this database contains arguments to the **automount** command, rather than mappings, and that **automount** does *not* look for an **auto.master** file on the local host.)

## OPTIONS

**–m** Suppress initialization of *directory-mapname* pairs listed in the **auto.master** Yellow Pages database.

**–n** Disable dynamic mounts. With this option, references through the **automount** daemon only succeed when the target filesystem has been previously mounted. This can be used to prevent NFS servers from cross-mounting each other.

**-T** Trace. Expand each NFS call and display it on the standard output.

**–tl** *duration*
Specify a *duration*, in seconds, that a looked up name remains cached when not in use. The default is 5 minutes.

**–tm** *interval*
Specify an *interval*, in seconds, between attempts to mount a filesystem. The default is 30 seconds.

**–tw** *interval*
Specify an *interval*, in seconds, between attempts to dismount filesystems that have exceeded their cached times. The default is 1 minute.

## EXAMPLE

**tutorial# automount –m /net –hosts**

Provide **automount** access to the exported file systems of any host in the Yellow Pages **hosts.byname** database, by prefixing the pathname with **/net/***hostname***/ :**

**tutorial% ls /net/hermes/usr/src ...**

## FILES

**/tmp_mnt** directory under which filesystems are dynamically mounted

## SEE ALSO

**mount**(8)

BUGS

Shell filename expansion does not apply to objects not currently mounted or cached. For instance, in the above example, the command **ls /net/*** might not list **hermes** as a subdirectory of **/net**.

## NAME

boot – start the system kernel, or a standalone program

## SYNOPSIS

**>b** [ *device* [ (*c*, *u*, *p*) ] ] [ *filename* ] [ **–a** ] *boot-flags*
**>b?**
**>b!**

## DESCRIPTION

The boot program is started by the PROM monitor and loads the kernel, or another executable program, into memory.

The form **b?** displays all boot devices and their device arguments.

The form **b!** boots, but does not perform a RESET.

## USAGE

### Booting Standalone

When booting standalone, the boot program (**/boot**) is brought in by the PROM from the file system. This program contains drivers for all devices.

### Booting a Sun-3 System Over the Network

When booting over the network, the Sun-3 system PROM obtains a version of the boot program from a server using the Trivial File Transfer Protocol (TFTP). The client broadcasts a RARP request containing its Ethernet address. A server responds with the client's Internet address. The client then sends a TFTP request for its boot program to that server (or if that fails, it broadcasts the request). The filename requested (unqualified — not a pathname) is the hexadecimal, uppercase representation of the client's Internet address, for example:

**Using IP Address 192.9.1.17 = C0090111**

When the Sun server receives the request, it looks in the directory **/tftpboot** for *filename*. That file is typically a symbolic link to the client's boot program, normally **boot.sun3** in the same directory. The server invokes the TFTP server, **tftpd**(8C), to transfer the file to the client.

When the file is successfully read in by the client, the boot program jumps to the load-point and loads **vmunix** (or a standalone program). In order to do this, the boot program makes a broadcast RARP request to find the client's IP address, and then makes a second broadcast request to a **bootparamd**(8) bootparams daemon, for information necessary to boot the client. The bootparams daemon obtains this information either from a local **/etc/bootparams** database file, or from a Yellow Pages (YP) map. The boot program sends two requests to the bootparams daemon, the first, **whoami**, to obtain its hostname, and the second, **getfile**, to obtain the name of the client's server and the pathname of the client's root partition.

The boot program then performs a **mount**(8) operation to mount the client's root partition, after which it can read in and execute any program within that partition by pathname (including a symbolic link to another file within that same partition). Typically, it reads in the file **/vmunix**. If the program is not read in successfully, **boot** responds with a short diagnostic message.

### Booting a Sun-2, Sun-4, or Sun386i System Over the Network

Sun-2, Sun-4 and Sun386i systems boot over the network in a similar fashion. However, the filename requested from a server must have a suffix that reflects the system architecture of the machine being booted. For these systems, the requested filename has the form:

*ip-address.arch*

where *ip-address* is the machine's Internet Protocol (IP) address in hex, and *arch* is a suffix representing its architecture. (Only Sun-3 systems may omit the *arch* suffix.) These filenames are restricted to 14 characters for compatibility with System V and other operating systems. Therefore, the architecture suffix is limited to 5 characters; it must be in upper case. At present, the following suffixes are recognized: **SUN2** for Sun-2 system, **SUN3** for Sun-3 system, **SUN2** for Sun-4 system, **S386** for Sun386i system, and **PCNFS** for PC-NFS.

Note: a Sun-2 system boots from its server using one extra step. It broadcasts an ND request which is intercepted by the user-level **ndbootd**(8) server. This server sends back a standalone program that carries out the same TFTP request sequence as is done for all the other systems.

**System Startup**

Once the system is loaded and running, the kernel performs some internal housekeeping, configures its device drivers, and allocates its internal tables and buffers. The kernel then starts process number 1 to run **init**(8), which performs file system housekeeping, starts system daemons, initializes the system console, and begins multiuser operation. Some of these activities are omitted when **init** is invoked with certain *boot-flags*. These are typically entered as arguments to the boot command, and passed along by the kernel to **init**.

OPTIONS

*device* One of:

| | |
|---|---|
| **ie** | Intel Ethernet |
| **ec** | 3Com Ethernet |
| **le** | Lance Ethernet (Sun 3-50 system) |
| **sd** | SCSI disk |
| **st** | SCSI 1/4" tape |
| **mt** | Tape Master 9-track 1/2" tape |
| **xt** | Xylogics 1/2" tape |
| **xy** | Xylogics 440/450 disk |

*c* Controller number, **0** if there is only one controller for the indicated type of device.

*u* Unit number, **0** if only there is only one driver.

*filename* Name of a standalone program in the selected partition, such as **stand/diag** or **vmunix**. Note: *filename* is relative to the root of the selected device and partition. It never begins with '/' (backslash). If *filename* is not given, the boot program uses a default value (normally **vmunix**). This is stored in the **vmunix** variable in the **boot** executable file supplied by Sun, but can be patched to indicate another standalone program loaded using **adb**(1).

**–a** Prompt interactively for the device and name of the file to boot. For more information on how to boot from a specific device, refer to *Installing the SunOS*.

*boot-flags* The boot program passes all *boot-flags* to the kernel or standalone program. They are typically arguments to that program or, as with those listed below, arguments to progams that it invokes.

**–b** Pass the **–b** flag through the kernel to **init**(8) to skip execution of the **/etc/rc.local** script.

**–h** Halt after loading the system.

**–s** Pass the **–s** flag through the kernel to **init**(8) for single-user operation.

**–i** *initname*
Pass the **–i** *initname* to the kernel to tell it to run *initname* as the first program rather than the default **/single/init**.

FILES

| | |
|---|---|
| **/boot** | standalone boot program |
| **/tftpboot/????????** | symbolic link to the boot program for a client |
| **/tftpboot/boot.sun3** | programs to boot from the client's root partition |
| **/usr/etc/in.tftpd** | TFTP server |
| **/usr/mdec/installboot** | program to install boot blocks from a remote host |
| **/vmunix** | |
| **/usr/boot** | |
| **/etc/bootparams** | |

**SEE ALSO**

**adb**(1), **tftp**(1), **bootparamd**(8), **init**(8), **mount**(8), **ndbootd**(8C), **rc**(8), **reboot**(8), **tftpd**(8C), **kadb**(8S), **monitor**(8S)

*Installing the SunOS*
*System and Network Administration*

**BUGS**

On the Sun-2 system, the PROM passes in the default name **vmunix**, overriding the the boot program's patchable default.

## NAME

bootparamd – boot parameter server

## SYNOPSIS

**/usr/etc/rpc.bootparamd** [ **–d** ]

## DESCRIPTION

**bootparamd** is a server process that provides information to diskless clients necessary for booting. It consults either the **bootparams** database or the **/etc/bootparams** file if the Yellow Pages service is not running.

**bootparamd** can be invoked either by **inetd**(8C) or by the user.

## OPTIONS

**–d** Display the debugging information.

## FILES

**/etc/bootparams**

## SEE ALSO

**inetd**(8C)

## NAME

C2conv, C2unconv – convert system to or from C2 security

## SYNOPSIS

**C2conv**

**C2unconv**

## AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**C2conv** converts a standard SunOS system to operate with C2-level security.

The program prompts for information regarding installation base, client systems (if the system is a SunDisk server), audit devices (if it is an audit file server), and names of file systems (if it is a remote audit server). The program also requests certain information for the **audit_control**(5) file; default values may be used for audit flags and for the "minfree" value. Finally, it requests the user ID of person (or list of persons) to notify (by **mail**(1)) when C2 administrative tasks are required. The default ID is **root** for the host being converted.

Once it has this information, **C2conv** uses it to set up the necessary files for a C2 secure system, reporting on its progress as it proceeds.

*C2unconv* backs out the changes made to **/etc/passwd** and **/etc/group**. It does not back out changes to other files.

## FILES

**/etc/passwd**
**/etc/group**
**/etc/fstab**

## SEE ALSO

**audit_control**(5)

NAME

captoinfo – convert a termcap description into a terminfo description

SYNOPSIS

**captoinfo** [ **–v** . . . ] [**–V**] [**–1**] [**–w** *width* ] *filename*. . .

DESCRIPTION

**captoinfo** converts the **termcap**(5) the terminal description entries given in *filename* into **terminfo**(5V) source entries, and writes them to the standard output along with any comments found in that file. A description that is expressed as relative to another description (as specified in the **termcap** *tc*= capability) is reduced to the minimum superset before being written.

If no *filename* is given, then the environment variable **TERMCAP** is used for the filename or entry. If **TERMCAP** is a full pathname to a file, only the terminal-name is specified in the environment variable **TERM** is extracted from that file. If that environment variable is not set, then the file **/etc/termcap** is read.

OPTIONS

**–v** Verbose. Print tracing information on the standard error as the program runs. Additional **–v** options increase the level of detail.

**–V** Version. Display the version of the program on the standard error and exit.

**–1** Print fields one-per-line. Otherwise, fields are printed several to a line, to a maximum width of 60 characters.

**–w** *width*

Change the output to *width* characters.

CAVEATS

Certain **termcap** defaults are assumed to be true. The bell character (**terminfo bel**) is assumed to be ^G. The linefeed capability (**termcap nl**) is assumed to be the same for both **cursor_down** and **scroll_forward** (**terminfo cud1** and **ind**, respectively.) Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for **termcap** fields such as **cursor_position** (**termcap cm**, **terminfo cup**) can sometimes produce a string that may not be optimal. In particular, the rarely used **termcap** operation *%n* produces strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a **termcap** entry, a hold-over from an earlier version of the system, has been removed.

FILES

**/usr/share/lib/terminfo/?/***

compiled terminal description database

**/etc/termcap**

SEE ALSO

**curses**(3V), **termcap**(5), **terminfo**(5V), **infocmp**(8V), **tic**(8V)

DIAGNOSTICS

**tgetent failed with return code n**

The termcap entry is not valid. In particular, check for an invalid '**tc=**' entry.

**unknown type given for the termcap code** *cc*.

The termcap description had an entry for *cc* whose type was not boolean, numeric or string.

**wrong type given for the boolean (numeric, string) termcap code** *cc*.

The boolean **termcap** entry *cc* was entered as a numeric or string capability.

**the boolean (numeric, string) termcap code** *cc* **is not a valid name.**

An unknown **termcap** code was specified.

**tgetent failed on TERM=term.**

The terminal type specified could not be found in the **termcap** file.

**TERM=term: cap** *cc* **(info** *ii***) is**
The **termcap** code was specified as a null string. The correct way to cancel an entry is with an '**@**', as in '**:bs@:**'. Giving a null string could cause incorrect assumptions to be made by the software which uses **termcap** or **terminfo**.

**a function key for** *cc* **was specified, but it already has the value**
*vv*. When parsing the **ko** capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

**the unknown termcap name** *cc* **was specified in the ko termcap capability.**
A key was specified in the **ko** capability which could not be handled.

**the** *vi* **character v (info** *ii***) has the value** *xx*, **but ma gives n.**
The **ma** capability specified a function key with a value different from that specified in another setting of the same key.

**the unknown** *vi* **key v was specified in the ma termcap capability.**
A vi(1) key unknown to **captoinfo** was specified in the **ma** capability.

**Warning: termcap sg (***nn***) and termcap ug (***nn***) had different values.**
**terminfo** assumes that the **sg** (now **xmc**) and **ug** values were the same.

**Warning: the string produced for** *ii* **may be inefficient.**
The parameterized string being created should be rewritten by hand.

**Null termname given.**
The terminal type was null. This is given if the environment variable **TERM** is not set or is null.

**cannot open** *filename* **for reading.**
The specified file could not be opened.

NAME

catman – create the cat files for the manual

SYNOPSIS

**/usr/etc/catman** [ **–nptw** ] [ **–M directory** ] [ **–T** *tmac.an* ] [ *sections* ]

DESCRIPTION

**catman** creates the preformatted versions of the on-line manual from the **nroff(1)** input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, **catman** recreates the **whatis** database.

If there is one parameter not starting with a '–', it is taken to be a list of manual sections to look in. For example

**catman 123**

only updates manual sections **1, 2,** and **3.**

If an unformatted source file contains only a line of the form '**.so manx/yyy.x**', a symbolic link is made in the catx or fmtx directory to the appropriate preformatted manual page. This feature allows easy distribution of the preformatted manual pages among a group of associated machines with **rdist**(1), since it makes the directories of preformatted manual pages self-contained and independent of the unformatted entries.

OPTIONS

**–n** Do not (re)create the **whatis** database.

**–p** Print what would be done instead of doing it.

**–t** Create **troff**ed entries in the appropriate **fmt** subdirectories instead of **nroff**ing into the **cat** subdirectories.

**–w** Only create the **whatis** database. No manual reformatting is done.

**–M** Update manual pages located in the specified **directory** (**/usr/share/man** by default).

**–T** Use **tmac.an** in place of the standard manual page macros.

ENVIRONMENT

**TROFF** The name of the formatter to use when the **–t** flag is given. If not set, '**troff**' is used.

FILES

**/usr/share/man** default manual directory location

**/usr/share/man/man?/*.*** raw (nroff input) manual sections

**/usr/share/man/cat?/*.*** preformatted **nroff**ed manual pages

**/usr/share/man/fmt?/*.*** preformatted **troff**ed manual pages

**/usr/share/man/whatis** **whatis** database location

**/usr/lib/makewhatis** command script to make whatis database

SEE ALSO

**man**(1), **nroff**(1), **rdist**(1), **troff**(1), **whatis**(1)

DIAGNOSTICS

**man?/xxx.? (.so'ed from man?/yyy.?): No such file or directory**

The file outside the parentheses is missing, and is referred to by the file inside them.

**target of .so in man?/xxx.? must be relative to /usr/man**

**catman** only allows references to filenames that are relative to the directory **/usr/share/man.**

**opendir:man?: No such file or directory**
A harmless warning message indicating that one of the directories **catman** normally looks for is missing.

***.*: No such file or directory**
A harmless warning message indicating **catman** came across an empty directory.

## NAME

chown – change owner

## SYNOPSIS

**/usr/etc/chown** [ **–fR** ] *owner*[*.group*] *filename* ...

## DESCRIPTION

**chown** changes the owner of the *filename*s to *owner*. The owner may be either a decimal user ID or a login name found in the password file. An optional *group* may also be specified. The group may be either a decimal group ID or a group name found in the GID file.

Only the super-user can change owner, in order to simplify accounting procedures.

## OPTIONS

**–f** Do not report errors.

**–R** Recursively descend into directories setting the ownership of all files in each directory encountered. When symbolic links are encountered, their ownership is changed, but they are not traversed.

## FILES

**/etc/passwd** password file

## SEE ALSO

**chgrp**(1), **chown**(2), **group**(5), **passwd**(5)

**NAME**

chroot – change root directory for a command

**SYNOPSIS**

**/usr/etc/chroot** *newroot command*

**DESCRIPTION**

The given command is executed *relative* to the new root. The meaning of any initial '/' (slashes) in path names is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Input and output redirections on the command line are made with respect to the *original* root:

**chroot newroot command >x**

creates the file *x* relative to the original root, not the new one.

This command is restricted to the super-user.

The new root path name is always relative to the current root: even if a **chroot** is already in effect; the *newroot* argument is relative to the current root of the running process.

**SEE ALSO**

**chdir**(2)

**BUGS**

One should exercise extreme caution when referring to special files in the new root file system.

NAME

client – add or remove diskless Sun386i systems

SYNOPSIS

**client** [ **–a** *arch* ] [ **–h** *hostid* ] [ **–o** *os* ] [ **–q** ] [ **–t** *minutes* ] **add** *bootserver client etheraddress ipaddress*

**client remove** *client*

**client modify** *client* [ **diskful** | **diskless** | **slave** ]

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**client** can be used to manually add and remove diskless clients of a PNP boot server. After successful completion of the command, the diskless client can boot. Only users in the *networks* group (group 12) on the boot server are allowed to change configurations using this utility. **client** can be invoked from any system on the network.

The boot server of a system is the only machine truly required for that system to boot to the point of allowing user logins; it must accordingly provide name, booting, and time services. Diskless clients can provide none of these services themselves. Diskful clients, however can provide most of their own boot services. Network clients only need name and time services from the network, and can use any boot server.

To add a diskless client, use the **add** operation. To remove a diskless, diskful, or network client, use the **remove** operation. To change a system's network role, use the **modify** operation.

A server can reject a configuration request if it is disallowed by the contents of the **bootservers** map (e.g., too many clients would be configured, or too little free space would be left on the server), or if no system software for the client is available.

OPTIONS

**–a** *arch* Specifies the architecture code of the client; it defaults to **s386**. (Note: architecture codes are different from architecture names. Architecture codes are used in diskless booting, and are at most five characters in length, while architecture names can be longer.)

**–h** *hostid* Specifies the host ID of the client; if supplied, it is used as the root password for the system. It defaults to the null string.

**–o** *os* Specifies the operating system; defaults to 'unix'. This is currently used only to construct the system's *publickey* data, where applicable; this is never done if the system has no *hostid* specified.

**–q** Quiet. Displays only error messages.

**–t** *minutes* Sets the RPC timeout to the number of minutes indicated; this defaults to 15 minutes. If the bootserver takes more time than this to complete, **client** will exit. Unless the server has already completed setup, but not yet sent status to **client**, this will cause the bootserver to back out of the setup, deallocating all assigned resources.

SEE ALSO

**pnpd**(8C), **netconfig**(8C), **publickey**(5)

BUGS

Unless the *hostid* is assigned, the root filesystem for the diskless client is not set up beyond copying the **proto** and **boot** files into it. This means that **netconfig** will often handle other parts of the setup.

**NAME**

clri – clear inode

**SYNOPSIS**

**/usr/etc/clri** *filesystem i-number...*

**DESCRIPTION**

Note: **clri** has been superceded for normal file system repair work by **fsck**(8).

**clri** writes zeros on the inodes with the decimal *i-numbers* on the *filesystem*. After **clri,** any blocks in the affected file will show up as "missing" in an **icheck**(8) of the *filesystem.*

Read and write permission is required on the specified file system device. The inode becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an inode which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated inode, so the whole cycle is likely to be repeated again and again.

**SEE ALSO**

**icheck**(8) **fsck**(8)

**BUGS**

If the file is open, **clri** is likely to be ineffective.

## NAME

comsat – biff server

## SYNOPSIS

**/usr/etc/in.comsat**

## DESCRIPTION

**comsat** is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by **inetd**(8C), and times out if inactive for a few minutes.

**comsat** listens on a datagram port associated with the **biff**(1) service specification (see **services**(5)) for one line messages of the form

**user@mailbox-offset**

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a '**biff y**'), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the **From, To, Date,** or **Subject** lines are not printed when displaying the message.

## FILES

| | |
|---|---|
| **/etc/utmp** | to find out who's logged on and on what terminals |

## SEE ALSO

**biff**(1), **services**(5), **inetd**(8C)

## BUGS

The message header filtering is prone to error.

The notification should appear in a separate window so it does not mess up the screen.

## NAME

config – build system configuration files

## SYNOPSIS

**/etc/config** [ **–fgnp** ] [ **–o** *obj_dir* ] *config_file*

## DESCRIPTION

**config** does the preparation necessary for building a new system kernel with **make**(1). The *config_file* named on the command line describes the kernel to be made in terms of options you want in your system, size of tables, and device drivers to be included. When you run **config**, it uses several input files located in the current directory (typically the **conf** subdirectory of the system source including your *config_file*). The format of this file is described below.

If the directory named *../config_file* does not exist, **config** will create one. One of **config**'s output files is a makefile which you use with **make**(1) to build your system.

You use **config** as follows. Run **config** from the **conf** subdirectory of the system source (in a typical Sun environment, from **/usr/include/sys/conf**):

**example# /etc/config config_file**
**Doing a "make depend"**
**example# cd ../config_file**
**example# make**
*... lots of output...*

While **config** is running watch for any errors. Never use a kernel which **config** has complained about; the results are unpredictable. If **config** completes successfully, you can change directory to the *../config_file* directory, where it has placed the new makefile, and use **make** to build a kernel. The output files placed in this directory include **ioconf.c**, which contains a description of I/O devices attached to the system; **mbglue.s**, which contains short assembly language routines used for vectored interrupts, a makefile, which is used by **make** to build the system; a set of header files (*device_name*.**h**) which contain the number of various devices that may be compiled into the system; and a set of swap configuration files which contain definitions for the disk areas to be used for the root file system, swapping, and system dumps.

Now you can install your new kernel and try it out.

## OPTIONS

**–f** Set up the makefile for fast builds. This is done by building a **vmunix.o** file which includes all the **.o** files which have no source. This reduces the number of files which have to be **stat**ed during a system build. This is done by prelinking all the files for which no source exists into another file which is then linked in place of all these files when the kernel is made. This makefile is faster because it does not **stat** the object files during the build.

**–g** Get the current version of a missing source file from its SCCS history, if possible.

**–n** Do not do the '**make depend**'. Normally **config** will do the '**make depend**' automatically. If this option is used **config** will print '**Don't forget to do a "make depend"**' before completing as a reminder.

**–p** Configure the system for profiling (see **kgmon**(8) and **gprof**(1)).

**–o** *obj_dir*
Use *../obj_dir* instead of *../***OBJ** as the directory to find the object files when the corresponding source file is not present in order to generate the files necessary to compile and link your kernel.

## USAGE

### Input Grammar

In the following descriptions, a number can be a decimal integer, a whole octal number or a whole hexadecimal number. Hex and octal numbers are specified to **config** in the same way they are specified to the C compiler, a number starting with **0x** is a hex number and a number starting with just a **0** is an octal number.

Comments are begin with a # character, and end at the next NEWLINE. Lines beginning with TAB characters are considered continuations of the previous line. Lines of the configuration file can be one of two basic types. First, there are lines which describe general things about your system:

**machine "***type***"**

This is system is to run on the machine type specified. Only one machine type can appear in the config file. The legal *type*s for a Sun system are **sun2, sun3, sun4,** and **sun386.** Note: the double quotes around *type* are part of the syntax, and must be included.

**cpu "***type***"**

This system is to run on the cpu type specified. More than one cpu type can appear in the config file. Legal *type*s for a **sun2** machine are noted in the annotated config file in *Installing the SunOS.*

**ident** *name*

Give the system identifier — a name for the machine or machines that run this kernel. Note that *name* must be enclosed in double quotes if it contains both letters and digits. Also, note that if *name* is **GENERIC,** you need not include the '**options GENERIC**' clause in order to specify '**swap generic**'.

**maxusers** *number*

The maximum expected number of simultaneously active user on this system is *number*. This number is used to size several system data structures.

**options** *optlist*

Compile the listed options into the system. Options in this list are separated by commas. A line of the form:

**options FUNNY, HAHA**

yields

**–DFUNNY –DHAHA**

to the C compiler. An option may be given a value, by following its name with = (equal sign) then the value enclosed in (double) quotes. None of the standard options use such a value.

In addition, options can be used to bring in additional files if the option is listed in the **files** files. All options should be listed in upper case. In this case, no corresponding *option*.**h** will be created as it would be using the corresponding *pseudo-device* method.

**config** *sysname config_clauses...*

Generate a system with name *sysname* and configuration as specified in *config-clauses*. The *sysname* is used to name the resultant binary image and per-system swap configuration files. The *config_clauses* indicate the location for the root file system, one or more disk partitions for swapping and paging, and a disk partition to which system dumps should be made. All but the root device specification may be omitted; **config** will assign default values as described below.

**root** A root device specification is of the form '**root on** *xy0d*'. If a specific partition is omitted — for example, if only **root on xy0** is specified — the '**a**' partition is assumed. When a generic system is being built, no root specification should be given; the root device will be defined at boot time by prompting the console.

**swap** To specify a swap partition, use a clause of the form: '**swap on** *partition*'. Swapping areas may be almost any size. Partitions used for swapping are sized at boot time by the system; to override dynamic sizing of a swap area the number of sectors in the swap area can be specified in the config file. For example, '**swap on** *xy0b* **size 99999**' would configure a swap partition with 99999 sectors. If **swap generic** or no *partition* is specified with **on**, partition b on the root device is used. For dataless clients, use '**swap on type nfs**'.

**dumps** The location to which system dumps are sent may be specified with a clause of the form '**dumps on** *xyl*'. If no dump device is specified, the first swap partition specified is used. If a device is specified without a particular partition, the '**b**' partition is assumed. If a generic configuration is to be built, no dump device should be specified; the dump device will be assigned to the swap device dynamically configured at boot time. Dumps are placed at the end of the partition specified. Their size and location is recorded in global kernel variables *dumpsize* and *dumplo*, respectively, for use by **savecore**(8).

Device names specified in configuration clauses are mapped to block device major numbers with the file **devices.***machine*, where *machine* is the machine type previously specified in the configuration file. If a device name to block device major number mapping must be overridden, a device specification may be given in the form '**major** *x* **minor** *y*'.

The second group of lines in the configuration file describe which devices your system has and what they are connected to (for example, a Xylogics 450 Disk Controller at address 0xee40 in the Multibus I/O space). These lines have the following format:

*dev_type dev_name* **at** *con_dev more_info*

*dev_type* is either **controller, disk, tape, device,** or **pseudo-device.** These types have the following meanings:

| | |
|---|---|
| **controller** | A disk or tape controller. |
| **disk** or **tape** | Devices connected to a controller. |
| **device** | Something "attached" to the main system bus, like a cartridge tape interface. |
| **pseudo-device** | A software subsystem or driver treated like a device driver, but without any associated hardware. Current examples are the pseudo-tty driver and various network subsystems. For pseudo-devices, **more_info** may be specified as an integer, that gives the value of the symbol defined in the header file created for that device, and is generally used to indicate the number of instances of the pseudo-device to create. |

*dev_name* is the standard device name and unit number (if the device is not a **pseudo-device**) of the device you are specifying. For example, **xyc0** is the *dev_name* for the first Xylogics controller in a system; **ar0** names the first quarter-inch tape controller.

*con_dev* is what the device you are specifying is connected to. It is either nexus?, a bus type, or a controller. There are several bus types which are used by **config** and the kernel.

The different possible bus types are:

| | |
|---|---|
| **obmem** | On board memory |
| **obio** | On board io |
| **mbmem** | Multibus memory (**sun2** system only) |
| **mbio** | Multibus io (**sun2** system only) |
| **vme16d16 (vme16)** | 16 bit VMEbus/ 16 bit data |
| **vme24d16 (vme24)** | 24 bit VMEbus/ 16 bit data |
| **vme32d16** | 32 bit VMEbus/ 16 bit data (**sun3** system only) |
| **vme16d32** | 16 bit VMEbus/ 32 bit data (**sun3** system only) |
| **vme24d32** | 24 bit VMEbus/ 32 bit data (**sun3** system only) |
| **vme32d32 (vme32)** | 32 bit VMEbus/ 32 bit data (**sun3** system only) |

All of these bus types are declared to be connected to nexus. The devices are hung off these buses. If the bus is wildcarded, then the autoconfiguation code will determine if it is appropriate to probe for the device on the machine that it is running on. If the bus is numbered, then the autoconfiguation code will only look for that device on machine type N. In general, the Multibus and VMEbus bus types are always wildcarded.

*more_info* is a sequence of the following:

**csr** *address* — Specify the address of the **csr** (command and status registers) for a device. The **csr** addresses specified for the device are the addresses within the bus type specified.

The **csr** address must be specified for all controllers, and for all devices connected to a main system bus.

**drive** *number* — For a disk or tape, specify which drive this is.

**flags** *number* — These flags are made available to the device driver, and are usually read at system initialization time.

**priority** *level* — For devices which interrupt, specify the interrupt level at which the device operates.

**vector** *intr number* [ *intr number* . . . ]
For devices which use vectored interrupts on VMEbus systems, *intr* specify the vectored interrupt routine and *number* the corresponding vector to be used (0x40-0xFF).

A **?** may be substituted for a number in two places and the system will figure out what to fill in for the **?** when it boots. You can put question marks on a *con_dev* (for example, at virtual '?'), or on a drive number (for example, drive '?'). This allows redundancy, as a single system can be built which will boot on different hardware configurations.

The easiest way to understand **config** files it to look at a working one and modify it to suit your system. Good examples are provided in *Installing the SunOS*.

## FILES

Files in **/usr/include/sys/conf** which may be useful for developing the *config_file* used by **config** are:

**GENERIC** — These are generic configuration files for either a Sun-2 or Sun-3 system. They contain all possible device descriptions lines for the particular architecture.

**README** — File describing how to make a new kernel.

As shipped from Sun, the files used by **/etc/config** as input are in the **/usr/include/sys/conf** directory:

| | |
|---|---|
| *config_file* | System-specific configuration file |
| **makefile.sun[23]** | Generic prototype makefile for Sun–[23] systems |
| **files** | List of common files required to build a basic kernel |
| **files.sun[23]** | List of files for a Sun–[23] specific kernel |
| **devices.sun[23]** | Name to major device mapping file for Sun–[23] systems |

**/etc/config** places its output files in the *../config_file* directory:

| | |
|---|---|
| **mbglue.s** | Short assembly language routines used for vectored interrupts |
| **ioconf.c** | Describes I/O devices attached to the system |
| *makefile* | Used with **make**(1) to build the system |
| *device_name*.**h** | a set of header files (various *device_name*'s) containing devices which can be compiled into the system |

## SEE ALSO

**gprof**(1), **make**(1), **kgmon**(8), **savecore**(8)

The SYNOPSIS portion of each device entry in Section 4 of this manual.

*Installing the SunOS*
*System and Network Administration*

## NAME

crash – what happens when the system crashes

## DESCRIPTION

This section explains what happens when the system crashes and how you can analyze crash dumps.

When the system crashes voluntarily, it displays a message of the form

**panic:** *why i gave up the ghost*

on the console, takes a dump on a mass storage peripheral, and then invokes an automatic reboot procedure as described in **reboot**(8). Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure, the system will then resume multiuser operations.

The system has a large number of internal consistency checks; if one of these fails, it will panic with a very short message indicating which one failed.

When the system crashes it writes (or at least attempts to write) an image of memory into the back end of the primary swap area. After the system is rebooted, you can run the program **savecore**(8) to preserve a copy of this core image and kernel namelist for later perusal. See **savecore**(8) for details.

To analyze a dump you should begin by running **adb**(1) with the **–k** flag on the core dump, as described in *Debugging Tools*

The most common cause of system failures is hardware failure, which can reflect itself in different ways.

Here are some messages that you may encounter, with some hints as to causes. In each case there is a possibility that a hardware or software error produced the message in some unexpected way.

## FILES

| | |
|---|---|
| **/vmunix** | the system kernel |
| **/etc/rc.local** | script run when the local system starts up |

## SEE ALSO

**adb**(1), **analyze**(8), **reboot**(8) sa(8), **savecore**(8)

*Debugging Tools*

## DIAGNOSTICS

**IO err in push**

**hard IO err in swap** The system encountered an error trying to write to the paging device or an error in reading critical information from a disk drive. You should fix your disk if it is broken or unreliable.

**timeout table overflow**

This really should not be a panic, but until the data structure is fixed, involved, running out of entries causes a crash. If this happens, you should make the timeout table bigger by changing the value of **ncallout** in the **param.c** file, and then rebuild your system.

**trap type** *type*, **pid** *process-id*, **pc** = *program-counter*, **sr** = *status-register*, **context** *context-number*

A unexpected trap has occurred within the system; typical trap types are:

- Bus error
- Address error
- Illegal instruction
- Divide by zero
- Chk instruction
- Trapv instruction
- Privilege violation
- Trace
- 1010 emulator trap
- 1111 emulator trap
- Stack format error

- Uninitialized interrupt
- Spurious interrupt

The favorite trap types in system crashes are "Bus error" or "Address error", indicating a wild reference. The *process-id* is the ID of the process running at the time of the fault, *program-counter* is the hexadecimal value of the program counter, *status-register* is the hexadecimal value of the status register, and *context-number* is the context that the process was running in. These problems tend to be easy to track down if they are kernel bugs since the processor stops cold, but random flakiness seems to cause this sometimes.

**init died** The system initialization process has exited. This is bad news, as no new users will then be able to log in. Rebooting is the only fix, so the system just does it right away.

## NAME

cron – clock daemon

## SYNOPSIS

**/usr/etc/cron**

## DESCRIPTION

**cron** executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in **crontab** files in the directory **/var/spool/cron/crontabs**. Users can submit their own **crontab** files using the **crontab**(1) command. Commands that are to be executed only once may be submitted using the **at**(1) command.

**cron** only examines **crontab** files and **at** command files during process initialization and when a file changes using **crontab** or **at**. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since **cron** never exits, it should only be executed once. This is normally done by running **cron** from the initialization process through the file **/etc/rc**; see **init**(8). **/var/spool/cron/FIFO** is a FIFO file that **crontab** and **at** use to communicate with **cron**; it is also used as a lock file to prevent the execution of more than one **cron**.

## FILES

| | |
|---|---|
| **/var/spool/cron** | main cron directory |
| **/var/spool/cron/FIFO** | FIFO for sending messages to **cron** |
| **/var/spool/cron/crontabs** | directory containing **crontab** files |

## SEE ALSO

**at**(1), **crontab**(1), **sh**(1), **init**(8), **syslogd**(8)

## DIAGNOSTICS

**cron** logs various errors to the system log daemon, **syslogd**(8), with a facility code of **cron**. The messages are listed here, grouped by severity level.

Err Severity

**Can't create /var/spool/cron/FIFO:** *reason*
: **cron** was unable to start up because it could not create **/var/spool/cron/FIFO**.

**Can't access /var/spool/cron/FIFO:** *reason*
: **cron** was unable to start up because it could not access **/var/spool/cron/FIFO**.

**Can't open /var/spool/cron/FIFO:** *reason*
: **cron** was unable to start up because it could not open **/var/spool/cron/FIFO**.

**Can't start cron - another cron may be running (/var/spool/cron/FIFO exists)**
: **cron** found that **/var/spool/cron/FIFO** already existed when it was started; this normally means that **cron** had already been started, but it may mean that an earlier **cron** terminated abnormally without removing **/var/spool/cron/FIFO**.

**Can't stat /var/spool/cron/FIFO:** *reason*
: **cron** could not get the status of **/var/spool/cron/FIFO**.

**Can't change directory to** *directory*:*reason*
: **cron** could not change to the directory *directory*.

**Can't read** *directory*:*reason*
: **cron** could not read the directory *directory*.

**error reading message:** *reason*
: An error occurred when **cron** tried to read a control message from **/var/spool/cron/FIFO**.

**message received — bad format**
A message was successfully read by **cron** from **/var/spool/cron/FIFO**, but the message was not of a form recognized by **cron**.

**SIGTERM**
received **cron** was told to terminate by having a SIGTERM signal sent to it.

**cron could not unlink /var/spool/cron/FIFO:** *reason*
**cron** was told to terminate, but it was unable to unlink **/var/spool/cron/FIFO** before it terminated.

********* CRON ABORTED **********
**cron** terminated, either due to an error or because it was told to.

**Can't open queuedefs file** *file*:*reason*
**cron** could not open a *queuedefs* file.

**I/O error reading queuedefs file** *file*:*reason*
An I/O error occurred while **cron** was reading a *queuedefs* file.

**Using default queue definitions**
An error occurred while trying to read a *queuedefs* file; the default queue definitions will be used.

**Can't allocate** *number* **bytes of space**
An internal error occurred in **cron** while trying to allocate memory.

**Info Severity**

*queue* **queue max run limit reached**
There were more jobs running or to be run in the queue *queue* than the maximum number specified. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

**MAXRUN (25) procs reached**
There were more than 25 jobs running or to be run by **cron**. **cron** will wait until one of the currently-running jobs completes before starting to run a new one.

***** cron started *****
**cron** started running.

**> CMD:** *command*
A **cron** job was started. *Command* is the command to be run.

> *user pid queue time*
A **cron** job was started for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*.

< *user pid queue time status*
A **cron** job completed for user *user*, in queue *queue*, with process ID *pid*, at the date and time *time*. If the command terminated with a non-zero exit status or a signal, *status* indicates the exit status or signal.

**Notice Severity**

**Can't fork**
An attempt to **fork (2)** to run a new job failed; **cron** will attempt again after a 30-second delay.

**Warning Severity**

**Can't stat queuedefs file** *file*:*reason*
**cron** could not get the status of a **queuedefs** file in order to determine whether it has changed. **cron** will assume it has changed and will reread it.

## NAME

dcheck – file system directory consistency check

## SYNOPSIS

**/usr/etc/dcheck** [ –**i** *numbers* ] [ *filesystem* ]

## DESCRIPTION

Note: **dcheck** has been superceded for normal consistency checking by **fsck**(8).

**dcheck** reads the directories in a file system and compares the link-count in each inode with the number of directory entries by which it is referenced. If the file system is not specified, **dcheck** checks a set of default file systems.

**dcheck** is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

## OPTIONS

–**i** *numbers*

*numbers* is a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

## FILES

Default file systems vary with installation.

## SEE ALSO

**fs**(5), **fsck**(8), **clri**(8), **icheck**(8), **ncheck**(8)

## DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

## BUGS

Since **dcheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Inode numbers less than 2 are invalid.

## NAME

devnm – device name

## SYNOPSIS

**/etc/devnm** [ *name* ] ...

## DESCRIPTION

**devnm** identifies the special file associated with the mounted file system where each *name* argument resides. This command can be used to construct a mount table entry for the **root** file system.

## EXAMPLE

If **/usr** is mounted on **/dev/dsk/c1d0s2**, then the command:

**/etc/devnm /usr**

produces:

**/dev/dsk/c1d0s2 usr**

## FILES

**/dev/dsk/***
**/etc/mtab**

## SEE ALSO

**mount**(8), **fstab**(5)

## NAME

dkinfo – report information about a disk's geometry and partitioning

## SYNOPSIS

**/usr/etc/dkinfo** *disk* [ *partition* ]

## DESCRIPTION

**dkinfo** gives the total number of cylinders, heads, and sectors or tracks on the specified *disk*, and gives this information along with the starting cylinder for the specified *partition*. If no *partition* is specified on the command line, **dkinfo** reports on all partitions.

The *disk* specification here is a disk name of the form *xxn*, where *xx* is the controller device abbreviation (ip, xy, etc.) and *n* is the disk number. The *partition* specification is simply the letter used to identify that partition in the standard UNIX system nomenclature. For example, **'/usr/etc/dkinfo xy0'** reports on the first disk in a system controlled by a Xylogics controller; **'/usr/etc/dkinfo xy0g'** reports on the seventh partition of such a disk.

## EXAMPLE

A request for information on my local disk, an 84 MByte disk controlled by a Xylogics 450 controller, might look like this:

```
#/usr/etc/dkinfo xy0
xy0: Xylogics 450 controller at addr ee40, unit # 0
586  cylinders 7 heads 32 sectors/track
a: 15884 sectors (70 cyls, 6 tracks, 12 sectors)
starting cylinder 0
b: 33440 sectors (149 cyls, 2 tracks)
starting cylinder 71
c: 131264 sectors (586 cyls)
starting cylinder 0
d: No such device or address
e: No such device or address
f: No such device or address
g: 81760 sectors (365 cyls)
starting cylinder 221
h: No such device or address
#
```

## FILES

**/dev/r***xxnp*

## SEE ALSO

**dkio**(4S), **format**(8)

## NAME

dmesg – collect system diagnostic messages to form error log

## SYNOPSIS

**/usr/etc/dmesg** [ – ]

## DESCRIPTION

Note: **dmesg** is obsoleted by **syslogd**(8) for maintenance of the system error log.

**dmesg** looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed or logged by the system when errors occur. If the '–' flag is given, then **dmesg** computes (incrementally) the new messages since the last time it was run and places these on the standard output.

## FILES

**/var/adm/msgbuf** scratch file for memory of '–' option

## SEE ALSO

**syslogd**(8)

## NAME

dump, rdump – incremental file system dump

## SYNOPSIS

**/usr/etc/dump** [ *options* [ *arguments* ] ] *filesystem*

## DESCRIPTION

**dump** backs up all files in *filesystem*, or files changed after a certain date, to magnetic tape; on Sun386i systems, **dump** works on both magnetic tape and diskettes. *options* is a string that specifies **dump** options, as shown below. Any *arguments* supplied for specific options are given as subsequent words on the command line, in the same order as that of the *options* listed.

If no *options* are given, the default is **9u**.

## OPTIONS

**0–9** The "dump level." All files in the *filesystem* that have been modified since the last **dump** at a lower dump level are copied to the volume. For instance, if you did a "level 2" dump on Monday, followed by a "level 4" dump on Tuesday, a subsequent "level 3" dump on Wednesday would contain all files modified or added since the "level 2" (Monday) backup. A "level 0" dump copies the entire filesystem to the dump volume.

**b** *factor* Blocking factor. Specify the blocking factor for tape writes. The default is 20 blocks per write. Note: the blocking factor is specified in terms of 512 bytes blocks, for compatibility with **tar**(1). The default blocking factor for tapes of density 6250BPI and greater is 64. The default blocking factor for cartridge tapes (**c** option specified) is 126. The highest blocking factor available with most tape drives is 126.

**c** Cartridge. Use a cartridge instead of the standard half-inch reel. This sets the density to 1000BPI and the blocking factor to 126. The length is set to 425 feet. (This option is incompatible with the **d** option, unless you specify a density of 1000BPI with that option).

**d** *bpi* Tape density. The density of the tape, expressed in BPI, is taken from *bpi*. This is used to keep a running tab on the amount of tape used per reel. The default density is 1600 except for cartridge tape. Unless a higher density is specified explicitly, **dump** uses its default density — even if the tape drive is capable of higher-density operation (for instance, 6250BPI). Note: the density specified should correspond to the density of the tape device being used, or **dump** will not be able to handle end-of-tape properly. The **d** option is not compatible with the **D** option.

**D** Diskette. Specify diskette as the dump media.

**f** *dump-file*

Dump file. Use *dump-file* as the file to dump to, instead of **/dev/rmt8.** If *dump-file* is specified as '–', dump to the standard output. If the filename argument is of the form *machine:device*, dump to a remote machine. Since **dump** is normally run by *root*, the name of the local machine must appear in the **.rhosts** file of the remote machine. If the filename argument is of the form *user@machine:device*, **dump** will attempt to execute as the specified user on the remote machine. The specified user must have a **.rhosts** file on the remote machine that allows root from the local machine. If **dump** is called as **rdump,** the dump device defaults to **dumphost:/dev/rmt8.** To direct the output to a desired remote machine, set up an alias for **dumphost** in the file **/etc/hosts.**

**n** Notify. When this option is specified, if **dump** requires attention, it sends a terminal message (similar to **wall**(1)) to all operators in the "operator" group.

**s** *size* Specify the *size* of the volume being dumped to. When the specified size is reached, **dump** waits for you to change the volume. **dump** interprets the specified size as the length in feet for tapes, and cartridges and as the number of 1024 byte blocks for diskettes. The following are defaults:

| | |
|---|---|
| tape | 2300 feet |
| cartridge | 425 feet |
| diskette | 1422 blocks (Corresponds to a 1.44 Mb diskette, with one cylinder reserved for bad block information.) |

**t** *tracks* Specify the number of tracks for a cartridge tape. On all Sun-2 systems the default is 4 tracks, although some Sun-2 systems have 9 track drives. On all other machines the default is 9 tracks. The **t** option is not compatible with the **D** option.

**u** Update the dump record. Add an entry to the file **/etc/dumpdates,** for each filesystem successfully dumped that includes the filesystem name, date, and dump level. This file can be edited by the super-user.

**w** List the filesystems that need backing up. This information is gleaned from the files **/etc/dumpdates** and **/etc/fstab**. When the **w** option is used, all other options are ignored. After reporting, **dump** exits immediately.

**W** Like **w**, but includes all filesystems that appear in **/etc/dumpdates,** along with information about their most recent dump dates and levels. Filesystems that need backing up are highlighted.

FILES

| | |
|---|---|
| **/dev/rmt8** | default unit to dump to |
| **dumphost:/dev/rmt8** | default remote unit to dump to if called as **rdump** |
| **/etc/dumpdates** | dump date record |
| **/etc/fstab** | dump table: file systems and frequency |
| **/etc/group** | to find group *operator* |
| **/etc/hosts** | |

SEE ALSO

**tar**(1), **wall**(1), **dump**(5), **fstab**(5), **restore**(8), **shutdown**(8)

DIAGNOSTICS

While running, **dump** emits many verbose messages.

**Exit Codes**

**0** Normal exit.
**1** Startup errors encountered.
**3** Abort – no checkpoint attempted.

BUGS

Fewer than 32 read errors on the filesystem are ignored.

Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

It is recommended that incremental dumps also be performed with the system running in single-user mode.

**dump** does not support multi-file multi-volume tapes.

NOTES

**Operator Intervention**

**dump** requires operator intervention on these conditions: end of volume, end of dump, volume write error, volume open error or disk read error (if there are more than a threshold of 32). In addition to alerting all operators implied by the **n** option, **dump** interacts with the operator on **dump**'s control terminal at times when **dump** can no longer proceed, or if something is grossly wrong. All questions **dump** poses *must* be answered by typing **yes** or **no**, as appropriate.

Since backing up a disk can involve a lot of time and effort, **dump** checkpoints at the start of each volume. If writing that volume fails for some reason, **dump** will, with operator permission, restart itself from the checkpoint after a defective volume has been replaced.

**dump** reports periodically, and in verbose fashion. Each report includes estimates of the percentage of the dump completed and how long it will take to complete the dump.

**Suggested Dump Schedule**

It is vital to perform full, "level 0", dumps at regular intervals. When performing a full dump, bring the machine down to single-user mode using **shutdown**(8). While preparing for a full dump, it is a good idea to clean the tape drive and heads.

Incremental dumps allow for convenient backup and recovery on a more frequent basis of active files, with a minimum of media and time. However there are some tradeoffs. First, the interval between backups should be kept to a minimum (once a day at least). To guard against data loss as a result of a media failure (a rare, but possible occurrence), it is a good idea to capture active files on (at least) two sets of dump volumes. Another consideration is the desire to keep unnecessary duplication of files to a minimum to save both operator time and media storage. A third consideration is the ease with which a particular backed-up version of a file can be located and restored. The following four-week schedule offers a reasonable trade-off between these goals.

| | *Sun* | *Mon* | *Tue* | *Wed* | *Thu* | *Fri* |
|---|---|---|---|---|---|---|
| *Week 1:* | **Full** | **5** | **5** | **5** | **5** | **3** |
| *Week 2:* | | **5** | **5** | **5** | **5** | **3** |
| *Week 3:* | | **5** | **5** | **5** | **5** | **3** |
| *Week 4:* | | **5** | **5** | **5** | **5** | **3** |

Although the Tuesday — Friday incrementals contain "extra copies" of files from Monday, this scheme assures that any file modified during the week can be recovered from the previous day's incremental dump.

**Process Priority of dump**

**dump** uses multiple processes to allow it to read from the disk and write to the media concurrently. Due to the way it synchronizes between these processes, any attempt to run dump with a **nice** (process priority) of '–5' or better will likely make **dump** run *slower* instead of faster.

## NAME

dumpfs – dump file system information

## SYNOPSIS

**/usr/etc/dumpfs** *device*

## DESCRIPTION

**dumpfs** prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

## SEE ALSO

**fs**(5), **fsck**(8), **newfs**(8), **tunefs**(8)

## NAME

edquota – edit user quotas

## SYNOPSIS

**/usr/etc/edquota** [ **–p** *proto-user* ] *usernames...*

**/usr/etc/edquota –t**

## DESCRIPTION

**edquota** is a quota editor. One or more users may be specified on the command line. For each user a temporary file is created with an ASCII representation of the current disk quotas for that user and an editor is then invoked on the file. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, **edquota** reads the temporary file and modifies the binary quota files to reflect the changes made.

The editor invoked is **vi**(1) unless the **EDITOR** environment variable specifies otherwise.

Only the super-user may edit quotas. (In order for quotas to be established on a file system, the root directory of the file system must contain a file, owned by root, called **quotas**. See **quotaon**(8) for details.)

## OPTIONS

**–p** Duplicate the quotas of the prototypical user specified for each user specified. This is the normal mechanism used to initialize quotas for groups of users.

**–t** Edit the soft time limits for each file system. If the time limits are zero, the default time limits in **<ufs/quota.h>** are used. Time units of sec(onds), min(utes), hour(s), day(s), week(s), and month(s) are understood. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

## FILES

| | |
|---|---|
| **quotas** | quota file at the file system root |
| **/etc/mtab** | mounted file systems |

## SEE ALSO

**quota**(1), **vi**(1), **quotactl**(2), **quotacheck**(8), **quotaon**(8), **repquota**(8)

## BUGS

The format of the temporary file is inscrutable.

## NAME

eeprom – EEPROM display and load utility

## SYNOPSIS

**eeprom** [ **–i** ] [ **–** ] [ **–f** *filename* ] [ *field* [ =*value* ] ] ...

**eeprom** [ **–i** ] [ **–c** ] [ **–f** *filename* ]

## AVAILABILITY

Not available for Sun-2 systems.

## DESCRIPTION

**eeprom** displays or changes the values of fields in the EEPROM. It processes fields in the order given. When processing a *field* accompanied by a *value*, **eeprom** makes the indicated alteration to the EEPROM; otherwise it displays the *field*'s value. When given no field specifiers, **eeprom** displays the values of all EEPROM fields. A '–' flag specifies that fields and values are to be read from stdin (one *field* or *field=value* per line).

**eeprom** verifies the EEPROM checksums and complains if they are incorrect; if the **–i** flag is specified, erroneous checksums are ignored. If the **–c** flag is specified, all incorrect checksums are recomputed and corrected in the EEPROM.

## OPTIONS

**–i** Ignore bad checksums.

**–f** *filename*
Use *filename* as the EEPROM device.

**–c** Correct bad checksums.

**–** Read field names and values from stdin.

The field names and their possible values are:

| | |
|---|---|
| **hwupdate** | a valid date (including "today" and "now") |
| **memsize** | 8 bit integer (megabytes of memory on machine) |
| **memtest** | 8 bit integer (megabytes of memory to test) |
| **scrsize** | "1024x1024", "1152x900", "1600x1280", or "1440x1440" |
| **watchdog_reboot** | "true" or "false" |
| **default_boot** | "true" or "false" |
| **bootdev** | %c%c (%x,%x,%x) |
| **kbdtype** | 8 bit integer (0 for all Sun keyboards) |
| **keyclick** | "true" or "false" |
| **console** | "b&w" or "ttya" or "ttyb" or "color" |
| **custom_logo** | "true" or "false" |
| **banner** | banner string |
| **diagdev** | %c%c (%x,%x,%x) - diagnostic boot device |
| **diagpath** | diagnostic boot path |
| **ttya_no_rtsdtr** | "true" or "false" |
| **ttyb_no_rtsdtr** | "true" or "false" |
| **columns** | number of columns on screen (8-bit integer) |
| **rows** | number of rows on screen (8-bit integer) |

## FILES

**/dev/eeprom**

## SEE ALSO

**<mon/eeprom.h>**

**NAME**

etherd – Ethernet statistics server

**SYNOPSIS**

**/usr/etc/rpc.etherd** *interface*

**AVAILABILITY**

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**etherd** is a server which puts *interface* into promiscuous mode, and keeps summary statistics of all the packets received on that interface. It responds to RPC requests for the summary. You must be root to run **etherd.**

*interface* is a networking interface such as **ie0, ie1, ec0, ec1** and **le0.**

**traffic**(1C) displays the information obtained from **etherd** in graphical form.

**SEE ALSO**

**traffic**(1C)

## NAME

etherfind – find packets on Ethernet

## SYNOPSIS

**etherfind** [ **–nprtuvx** ] [ **–c** *count* ] [ **–i** *interface* ] *expression*

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**etherfind** prints out the headers of packets on the ethernet that match the boolean *expression*. When an internet packet is fragmented into more than one ethernet packet, all fragments except the first are marked with an asterisk. You must be root to invoke **etherfind**.

## OPTIONS

**–n** Do not convert host addresses and port numbers to names.

**–p** Normally, the selected interface is put into promiscuous mode, so that **etherfind** has access to all packets on the ethernet. However, when the **-p** flag is used, the interface will not go promiscuous.

**–r** RPC mode: treat each packet as an RPC message, printing the program and procedure numbers.

**–t** Timestamps: precede each packet listing with a time value in seconds and hundredths of seconds since the first packet.

**–u** Make the output line buffered.

**–v** Verbose mode: print out some of the fields of TCP and UDP packets.

**–x** Dump the header in hex, in addition to the line printed for each packet by default.

**–c** *count*

Exit after receiving *count* packets. This is sometimes useful for dumping a sample of ethernet traffic to a file for later analysis.

**–i** *interface*

**etherfind** listens on *interface*. The program **netstat**(8C) when invoked with the **-i** flag lists all the interfaces that a machine has.

*expression*

The syntax of of *expression* is similar to that used by **find**(1). Here are the allowable primaries.

**–dst** *destination*

True if the destination field of the packet is *destination*, which may be either an address or a name.

**–src** *source*

True if the source field of the packet is *source*, which may be either an address or a name.

**–between** *host1 host2*

True if either the source of the packet is *host1* and the destination *host2*, or the source is *host2* and the destination *host1*.

**–dstnet** *destination*

True if the destination field of the packet has a network part of *destination*, which may be either an address or a name.

**–srcnet** *source*

True if the source field of the packet has a network part of *source*, which may be either an address or a name.

**–srcport** *port*
True if the packet has a source port value of *port*. It must be either upd or tcp (see **tcp**(4P)), **udp**(4P)). The *port* can be a number or a name used in **/etc/services.**

**–dstport** *port*
True if the packet has a destination port value of *port*. The *port* can be a number or a name.

**–less** *length*
True if the packet has a length less than or equal to *length*.

**–greater** *length*
True if the packet has a length greater than or equal to *length*.

**–proto** *protocol*
True if the packet is an ip packet (see **ip**(4P)) of protocol type *protocol*. *Protocol* can be a number or one of the names **icmp, udp, nd,** or **tcp.**

**–byte** *byte op value*
True if byte number *byte* of the packet is in relation *op* to *value*. Legal values for *op* are **+, <, >, &,** and **|**. Thus **4=6** is true if the fourth byte of the packet has the value 6, and **20&0xf** is true if byte twenty has one of its four low order bits nonzero.

**–broadcast**
True if the packet is a broadcast packet.

**–arp** True if the packet is a arp packet (see **arp**(4P)).

**–rarp** True if the packet is a rarp packet.

**–ip** True if the packet is an ip packet.

The primaries may be combined using the following operators (in order of decreasing precedence):

A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

The negation of a primary ('**!**' is the unary *not* operator).

Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

Alternation of primaries ('**–o**' is the *or* operator).

**EXAMPLE**

To find all packets arriving at or departing from sundown

```
example% etherfind –src sundown –o –dst sundown
example%
```

**SEE ALSO**

**find**(1), **traffic**(1C), **arp**(4P), **ip**(4P), **nit**(4P) **tcp**(4P), **udp**(4P), **netstat**(8C)

**BUGS**

The syntax is painful.

**NAME**

exportfs – export and unexport directories to NFS clients

**SYNOPSIS**

**/usr/etc/exportfs** [ **–avu** ] [ **–o** *options* ] [ **directory** ]

**DESCRIPTION**

**exportfs** makes a local directory (or file) available for mounting over the network by NFS clients. It is normally invoked at boot time by the **/etc/rc.local** script, and uses information contained in the **/etc/exports** file to export a *directory* (which must be specified as a full pathname). The super-user can run **exportfs** at any time to alter the list or characteristics of exported directories. Directories that are currently exported are listed in the file **/etc/xtab**.

With no options or arguments, **exportfs** prints out the list of directories currently exported.

**OPTIONS**

**–a** All. Export all directories listed in **/etc/exports**, or if **–u** is specified, unexport all of the currently exported directories.

**–v** Verbose. Print each directory as it is exported or unexported.

**–u** Unexport the indicated directories.

**–i** Ignore the options in **/etc/exports**. Normally, **exportfs** will consult **/etc/exports** for the options associated with the exported directory.

**–o** *options*

Specify a comma-separated list of optional characteristics for the directory being exported. *options* can be selected from among:

**ro** Export the directory read-only. If not specified, the directory is exported read-write.

**rw=***hostname*[**:***hostname*]...

Export the directory read-mostly. Read-mostly means exported read-only to most machines, but read-write to those specified. If not specified, the directory is exported read-write to all.

**anon=***uid*

If a request comes from an unknown user, use *uid* as the effective user ID. Note: root users (uid 0) are always considered "unknown" by the NFS server, unless they are included in the "root" option below. The default value for this option is –2. Setting the value of ''anon'' to -1 disables anonymous access. Note that by default secure NFS accepts insecure requests as anonymous, and those wishing for extra security can disable this feature by setting ''anon'' to -1.

**root=***hostname*[**:***hostname*]...

Give root access only to the root users from a specified *hostname*. The default is for no hosts to be granted root access.

**access=***client*[**:***client*]...

Give mount access to each *client* listed. A *client* can either be a hostname, or a netgroup (see **netgroup**(5)). Each *client* in the list is first checked for in the **/etc/netgroup** database, and then the **/etc/hosts** database. The default value allows any machine to mount the given directory.

**secure** Require clients to use a more secure protocol when accessing the directory.

**FILES**

| | |
|---|---|
| **/etc/exports** | static export information |
| **/etc/xtab** | current state of exported directories |
| **/etc/netgroup** | |

**SEE ALSO**

**exports(5), netgroup(5)**

**WARNINGS**

You cannot export a directory that is either a parent- or a sub-directory of one that is currently exported and *within the same filesystem.* It would be illegal, for example, to export both **/usr** and **/usr/local** if both directories resided in the same disk partition.

## NAME

extract_unbundled – extract and execute unbundled-product installation scripts

## SYNOPSIS

**extract_unbundled** [ **–d***device* [ **–r***remote-host* ] ] [ **–DEFAULT** ]

## DESCRIPTION

**extract_unbundled** extracts and executes the installation scripts from release tapes for Sun unbundled software products. If no options are specifed, it prompts for input as to the tape device, or remote host-name from which to the software is to be installed. For information about installing a specific product, refer to the installation manual that accompanies that product.

## OPTIONS

**–d***device*

Install from the indicated tape drive, such as **st0**, **mt0** or **ar0**.

**–r***remote_host*

Install from the device given in the **–d** option on the indicated remote host.

**–DEFAULT**

Execute the installation script using all default values. Otherwise the installation script prompts for any optional values.

## NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

## SYNOPSIS

**/usr/etc/fastboot** [ *boot-options* ]

**/usr/etc/fasthalt** [ *halt-options* ]

## DESCRIPTION

**fastboot** and **fasthalt** are shell scripts that reboot and halt the system without checking the file systems. This is done by creating a file **/fastboot**, then invoking the **reboot**(8) program. The system startup script, **/etc/rc**, looks for this file and, if present, skips the normal invocation of **fsck**(8).

## FILES

**/usr/etc/fastboot**
**/etc/rc**

## SEE ALSO

**fsck**(8), **halt**(8), **init**(8), **rc**(8), **reboot**(8)

## NAME

fingerd – remote user information server

## SYNOPSIS

**/usr/etc/in.fingerd**

## DESCRIPTION

**fingerd** implements the server side of the Name/Finger protocol, specified in RFC 742. The Name/Finger protocol provides a remote interface to programs which display information on system status and individual users. The protocol imposes little structure on the format of the exchange between client and server. The client provides a single "command line" to the finger server which returns a printable reply.

**fingerd** waits for connections on TCP port 79. Once connected it reads a single command line terminated by a <RETURN-LINE-FEED> which is passed to **finger**(1). **fingerd** closes its connections as soon as the output is finished.

If the line is null (only a RETURN-LINEFEED is sent) then **finger** returns a "default" report that lists all people logged into the system at that moment.

If a user name is specified (for instance, eric<RETURN-LINE-FEED>) then the response lists more extended information for only that particular user, whether logged in or not. Allowable "names" in the command line include both "login names" and "user names". If a name is ambiguous, all possible derivations are returned.

## SEE ALSO

**finger**(1)

Harrenstien, Ken, *NAME/FINGER*, RFC 742, Network Information Center, SRI International, Menlo Park, Calif., December 1977.

## BUGS

Connecting directly to the server from a TIP or an equally narrow-minded TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation. **fingerd** should be taught to filter out IAC's and perhaps even respond negatively (IAC *will not)* to all option commands received.

## NAME

format – disk partitioning and maintenance utility

## SYNOPSIS

**format** [ **–f** *command-file* ] [ **–l** *log-file* ] [ **–x** *data-file* ] [ **–d** *disk-name* ] [ **–t** *disk_type* ] [ **–p** *partition-name* ] [ **–s** ] *diskname*...

## DESCRIPTION

**format** enables you to format, label, repair and analyze disks on your Sun computer. Unlike previous disk maintenance programs, **format** runs under SunOS. Because there are limitations to what can be done to the system disk while the system is running, **format** is also supported within the memory-resident system environment. For most applications, however, running **format** under SunOS is the more convenient approach.

If no *disk-list* is present, **format** uses the disk list defined in the data file specified with the **–x** option. If that option is omitted, the data file defaults to **format.dat** in the current directory, or else **/etc/format.dat**.

## OPTIONS

**–f** *command-file*

Take command input from *command-file* rather than the standard input. The file must contain commands that appear just as they would if they had been entered from the keyboard. With this option, **format** does not issue **continue?** prompts.

**–l** *log-file*

Log a transcript of the **format** session to the indicated *log-file*, including the standard input, the standard output and the standard error.

**–x** *data-file*

Use the disk list contained in *data-file*.

**–d** *disk_name*

Specify which disk should be made current upon entry into the program. The disk is specified by its logical name (for instance, - xy0). This can also be accomplished by specifying a single disk in the disk list.

**–t** *disk-type*

Specify the type of disk which is current upon entry into the program, A disk's type is specified by name in the data file. This option can only be used if a disk is being made current as described above.

**–p** *partition-name*

Specify the partition table for the disk which is current upon entry into the program. The table is specified by its name as defined in the data file. This option can only be used if a disk is being made current, and its type is either specified or available from the disk label.

**–s** Silent. Suppress all of the standard output. Error messages are still displayed. This is generally used in conjunction with the **–f** option.

## FILES

**/etc/format.dat** default data file

NAME

fparel - Sun FPA online reliability tests

SYNOPSIS

**fparel** [ −p*n* ] [ −v ]

DESCRIPTION

**fparel** is a command to execute the Sun FPA online confidence and reliability test program. **fparel** tests about 90% of the functions of the FPA board, and tests all FPA contexts not in use by other processes. **fparel** runs without disturbing other processes that may be using the FPA. **fparel** can only be run by the super-user.

After a successful pass, **fparel** writes

**time, date: Sun FPA Passed. The contexts tested are: 0, 1, ... 31**

to the file **/var/adm/diaglog**.

If a pass fails, **fparel** writes

**time, date: Sun FPA failed**

along with the test name and context number that failed, to the file **/var/adm/diaglog**. **fparel** then broadcasts the message

**time, date: Sun FPA failed, disabled, service required**

to all users of the system. Next, **fparel** causes the kernel to disable the FPA. Once the kernel disables the FPA, the system must be rebooted to make it accessible.

The file **/etc/rc.local** should contain an entry to cause **fparel** to be invoked upon reboot to be sure that the FPA remains unaccessible in cases where rebooting doesn't correct the problem. See **rc**(8).

The **crontab**(5) file for root should contain an entry indicating that **cron**(8) is to run **fparel** daily, such as:

**7 2 * * * /usr/etc/fpa/fparel**

which causes **fparel** to run at seven minutes past two, every day. See **cron**(8) and **crontab**(5) for details.

OPTIONS

−p*n* Perform *n* passes. Default is *n*=1. −**p0** means perform 2147483647 passes.

−v Run in verbose mode with detailed test results to the standard output.

FILES

| | |
|---|---|
| **/var/adm/diaglog** | Log of **fparel** diagnostics. |
| **/etc/rc.local** | |
| **/var/spool/cron/crontabs/root** | |
| **/usr/etc/fpa/*** | directory containing FPA microcode, data files, and loader |

SEE ALSO

**fpaversion**(8), **crontab**(5), **cron**(8), **rc**(8)

NAME

fpaversion, fpa_download – print FPA version, load microcode

SYNOPSIS

**fpaversion** [ **–hlqv** ] [ **–t** [ **cdhimprstvxCIMS** ] ]

DESCRIPTION

**fpaversion** performs various tests on the FPA (floating point accelerator). With no arguments, it prints the version number of the microcode and constants that are currently installed on **/dev/fpa**, and performs a quick test to ensure proper operation.

OPTIONS

**–h** Help. Print command-line summary.

**–l** Loop through tests infinitely.

**–q** Quiet output. Print out only error messages.

**–v** Verbose output.

**–t** Specify certain tests:

- **c** Command register format instructions.
- **d** Double precision format instructions.
- **h** Help. Print summary of test specifiers.
- **i** Imask register.
- **m** Mode register.
- **p** Simple pipe sequencing.
- **r** User registers for all contexts.
- **s** Single precision format instructions.
- **t** Status generation.
- **v** Print version number and date of microcode and constants.
- **x** Extended format instructions.
- **C** Check checksum for microcode, mapping RAM, and constant RAM.
- **M** Command register format matrix instructions.
- **S** Shadow registers.

FILES

**/dev/fpa** physical FPA device
**/usr/etc/fpa/fpamicro*** microcode binaries for specific versions
**/usr/etc/fpa/fpa_constants**
microcode data file
**/usr/etc/fpa/fpa_download**
microcode loader

SEE ALSO

**fparel**(8), **sysdiag**(8)

## NAME

fsck – file system consistency check and interactive repair

## SYNOPSIS

**/usr/etc/fsck –p** [ *filesystem* ... ]

**/usr/etc/fsck** [ **–b** block# ] [ **–w** ] [ **–y** ] [ **–n** ] [ *filesystem* ] ...

## DESCRIPTION

The first form of *fsck* preens a standard set of file systems or the specified file systems. It is normally used in the **/etc/rc** script during automatic reboot. In this case, *fsck* reads the table **/etc/fstab** to determine the file systems to check. It inspects disks in parallel, taking maximum advantage of I/O overlap to check the file systems as quickly as possible.

Normally, the root file system is checked in pass 1; other root-partition file systems are checked in pass 2. Small file systems on separate partitions are checked in pass 3, while larger ones are checked in passes 4 and 5.

Only partitions marked in **/etc/fstab** with a file system type of "4.2" and a non-zero pass number are checked.

*fsck* corrects innocuous inconsistencies such as: unreferenced inodes, too-large link counts in inodes, missing blocks in the free list, blocks appearing in the free list and also in files, or incorrect counts in the super block, automatically. It displays a message for each inconsistency corrected that identifies the nature of, and file system on which, the correction is to take place. After successfully correcting a file system, *fsck* prints the number of files on that file system, the number of used and free blocks, and the percentage of fragmentation.

If *fsck* encounters other inconsistencies that it cannot fix automatically, it exits with an abnormal return status (and the reboot fails).

If sent a QUIT signal, *fsck* will finish the file system checks, then exit with an abnormal return status that causes the automatic reboot to fail. This is useful when you wish to finish the file system checks, but do not want the machine to come up multiuser.

Without the **–p** option, *fsck* audits and interactively repairs inconsistent conditions on file systems. In this case, it asks for confirmation before attempting any corrections. Inconsistencies other than those mentioned above can often result in some loss of data. The amount and severity of data lost can be determined from the diagnostic output.

The default action for each correction is to wait for the operator to respond either **yes** or **no**. If the operator does not have write permission on the file system, *fsck* will default to a **–n** (no corrections) action.

If no file systems are given to *fsck* then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Incorrect directory sizes.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks, file pointing to unallocated inode, inode number out of range.
8. Super Block checks: more blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. If the **lost+found** directory does not exist, it is created. If there is insufficient space its size is increased.

A file system may be specified by giving the name of the cooked or raw device on which it resides, or by giving the name of its mount point. If the latter is given, *fsck* finds the name of the device on which the file system resides by looking in **/etc/fstab**.

Checking the raw device is almost always faster.

## OPTIONS

**–b** Use the block specified immediately after the flag as the super block for the file system. Block 32 is always an alternate super block.

**–w** Check writable file systems only.

**–y** Assume a **yes** response to all questions asked by *fsck*; this should be used with extreme caution, as it is a free license to continue, even after severe problems are encountered.

**–n** Assume a **no** response to all questions asked by *fsck*; do not open the file system for writing.

## FILES

/etc/fstab contains default list of file systems to check

## DIAGNOSTICS

The diagnostics produced by *fsck* are fully enumerated and explained in *System and Network Administration.*

## EXIT STATUS

**0** Either no errors detected or all errors were corrected.

**4** Root file system errors were corrected. The system must be rebooted.

**8** Some uncorrected errors exist on one or more of the file systems checked, there was a syntax error, or some other operational error occurred.

**12** A signal was caught during processing.

## SEE ALSO

fstab(5), fs(5), newfs(8), mkfs(8), crash(8S), reboot(8)

*System and Network Administration*

## BUGS

There should be some way to start a **fsck –p** at pass *n*.

NAME

fsirand – install random inode generation numbers

SYNOPSIS

**fsirand** [ **–p** ] *special*

DESCRIPTION

**fsirand** installs random inode generation numbers on all the inodes on device *special*, and also installs a filesystem ID in the superblock. This helps increase the security of filesystems exported by NFS.

**fsirand** must be used only on an unmounted filesystem that has been checked with **fsck**(8). The only exception is that it can be used on the root filesystem in single-user mode, if the system is immediately rebooted afterwords.

OPTIONS

**–p** Print out the generation numbers for all the inodes, but do not change the generation numbers.

SEE ALSO

**fsck**(8)

## NAME

ftpd – DARPA Internet File Transfer Protocol server

## SYNOPSIS

**/usr/etc/in.ftpd** [ **–dl** ] [ **–t***timeout* ] *host.socket*

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**ftpd** is the DARPA Internet File Transfer Protocol (FTP) server process. The server is invoked by the Internet daemon **inetd**(8C) each time a connection to the FTP service (see **services**(5)) is made, with the connection available as descriptor 0 and the host and socket the connection originated from (in hex and decimal respectively) as argument.

Inactive connections are timed out after 60 seconds.

If the **–d** option is specified, debugging information is logged to the system log daemon, **syslogd**(8).

If the **–l** option is specified, each FTP session is logged to **syslogd**.

The FTP server will timeout an inactive session after 15 minutes. If the **–t** option is specified, the inactivity timeout period will be set to *timeout*.

The FTP server currently supports the following FTP requests; case is not distinguished.

| Request | Description |
|---|---|
| **ABOR** | abort previous command |
| **ACCT** | specify account (ignored) |
| **ALLO** | allocate storage (vacuously) |
| **APPE** | append to a file |
| **CDUP** | change to parent of current working directory |
| **CWD** | change working directory |
| **DELE** | delete a file |
| **HELP** | give help information |
| **LIST** | give list files in a directory ("ls –lg") |
| **MKD** | make a directory |
| **MODE** | specify data transfer *mode* |
| **NLST** | give name list of files in directory ("ls") |
| **NOOP** | do nothing |
| **PASS** | specify password |
| **PASV** | prepare for server-to-server transfer |
| **PORT** | specify data connection port |
| **PWD** | print the current working directory |
| **QUIT** | terminate session |
| **RETR** | retrieve a file |
| **RMD** | remove a directory |
| **RNFR** | specify rename-from file name |
| **RNTO** | specify rename-to file name |

**STOR** store a file

**STOU** store a file with a unique name

**STRU** specify data transfer *structure*

**TYPE** specify data transfer *type*

**USER** specify user name

**XCUP** change to parent of current working directory

**XCWD** change working directory

**XMKD** make a directory

**XPWD** print the current working directory

**XRMD** remove a directory

The remaining FTP requests specified in RFC 959 are recognized, but not implemented.

The FTP server will abort an active file transfer only when the **ABOR** command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in RFC 959.

**ftpd** interprets file names according to the "globbing" conventions used by **csh**(1). This allows users to utilize the metacharacters '*** ? [] {}~**'.

**ftpd** authenticates users according to three rules.

1) The user name must be in the password data base, **/etc/passwd**, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) The user must have a standard shell returned by **getusershell**(3).

3) If the user name is "anonymous" or "ftp", an anonymous FTP account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, **ftpd** takes special measures to restrict the client's access privileges. The server performs a **chroot**(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended.

~.B ftp Make the home directory owned by "ftp" and unwritable by anyone.

**~ftp/bin** Make this directory owned by the super-user and unwritable by anyone. The program **ls**(1V) must be present to support the list commands. This program should have mode 111.

**~ftp/etc** Make this directory owned by the super-user and unwritable by anyone. The files **passwd**(5) and **group**(5) must be present for the **ls** command to work properly. These files should be mode 444.

**~ftp/pub** Make this directory mode 777 and owned by "ftp". Users should then place files which are to be accessible via the anonymous account in this directory.

## DIAGNOSTICS

**ftpd** logs various errors to the system log daemon, **syslogd** , with a facility code of **daemon**. The messages are listed here, grouped by severity level.

**Err Severity**

getpeername failed: *reason*
A **getpeername**(2) call failed.

getsockname failed: *reason*
A **getsockname**(2) call failed.

signal failed: *reason*
A **signal**(3) call failed.

setsockopt failed: *reason*
A **setsockopt** call (see **getsockopt**(2)) failed.

ioctl failed: *reason*
A **ioctl**(2) call failed.

*directory*: *reason*
**ftpd** did not have write permission on the directory *directory* in which a file was to be created by the **STOU** command.

**Info Severity**
These messages are logged only if the **-l** flag is specified.

FTPD: connection from *host* at *time*
A connection was made to **ftpd** from the host *host* at the date and time *time*.

FTPD: User *user* timed out after *timeout* seconds at *time*
The user *user* was logged out because they hadn't entered any commands after *timeout* seconds; the logout occurred at the date and time *time*.

**Debug Severity**
These messages are logged only if the **-d** flag is specified.

FTPD: command: *command*
A command line containing *command* was read from the FTP client.

lost connection
The FTP client dropped the connection.

<—— *replycode*
<—— *replycode*–
A reply was sent to the FTP client with the reply code *replycode*. The next message logged will include the message associated with the reply. If a – follows the reply code, the reply is continued on later lines.

## SEE ALSO

**ftp**(1C), **getsockopt**(2), **getusershell**(3), **syslogd**(8)

Postel, Jon, and Joyce Reynolds, *File Transfer Protocol (FTP)*, RFC 959, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

## BUGS

The anonymous account is inherently dangerous and should be avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

gettable – get DoD Internet format host table from a host

SYNOPSIS

**/usr/etc/gettable** *host*

DESCRIPTION

**gettable** is a simple program used to obtain the DoD Internet host table from a "hostname" server. The indicated *host* is queried for the table. The table, if retrieved, is placed in the file **hosts.txt**.

**gettable** operates by opening a TCP connection to the port indicated in the service specification for "hostname". A request is then made for "ALL" names and the resultant information is placed in the output file.

**gettable** is best used in conjunction with the **htable**(8) program which converts the DoD Internet host table format to that used by the network library lookup routines.

SEE ALSO

**intro**(3N), **htable**(8)

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *HOSTNAME Server*, RFC 953, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

BUGS

Should allow requests for only part of the database.

## NAME

getty – set terminal mode

## SYNOPSIS

**/usr/etc/getty** [ *type* [ *tty* ] ]

## Sun386i SYSTEM SYNOPSIS

**/usr/etc/getty** [ **–n** ] [ *type* [ *tty* ] ]

## DESCRIPTION

**getty**, which is invoked by **init**(8), opens and initializes a tty line, reads a login name, and invokes **login**(1).

The *tty* argument is the name of the character-special file in **/dev** that corresponds to the terminal. If there is no *tty* argument, or the argument is '–', the tty line is assumed to be opened as file descriptor **0**.

The *type* argument, if supplied, is used as an index into the **gettytab**(5) database—to determine the characteristics of the line. If this argument is absent, or if there is no such entry, the default entry is used. If there is no **/etc/gettytab** file, a set of system-supplied defaults is used.

When the indicated entry is located, **getty** clears the terminal screen, prints a banner heading, and prompts for a login name. Usually, either the banner or the login prompt includes the system's hostname.

Next, **getty** prompts for a login and reads the login name, one character at a time. When it receives a NULL character (which is assumed to be the result pressing the BREAK , or "interrupt" key), **getty** switches to the entry **gettytab** entry named in the **nx** field. It reinitializes the line to the new characteristics, and then prompts for a login once again. This mechanism typically is used to cycle through a set of line speeds (baud rates) for each terminal line. For instance, a rotary dialup might have entries for the speeds: 300, 1200, 150, and 110 baud, with each **nx** field pointing to the next one in succession.

The user terminates login input line with a NEWLINE or RETURN character. The latter is preferable; it sets up the proper treatment of RETURN characters (see **tty**(4)). **getty** checks to see if the terminal has only upper-case alphabetical characters. If all alphabetical characters in the login name are in upper case, the system maps them along with all subsequent upper-case input characters to lower-case internally; they are displayed in upper case for the benefit of the terminal. To force recognition of an upper-case character, the shell allows them to be quoted (typically by preceding each with a backslash, '\').

Finally, **getty** calls **login**(1) with the login name as an argument.

**getty** can be set to time out after a certain interval; this hangs up dial-up lines if the login name is not entered in time.

## Sun386i SYSTEM DESCRIPTION

For Sun386i system, the value of *type* is the constant **Sun**, for the console frame buffer.

## Sun386i SYSTEM OPTIONS

**–n** invoke the full screen login program **logintool**(8), and optionally the "New User Accounts" feature. May only be used on a frame buffer. Unless removed from the console entry in */etc/ttytab*, this option is in effect by default.

## FILES

**/etc/gettytab**

## SEE ALSO

**login**(1), **ioctl**(2), **tty**(4), **gettytab**(5), **ttytab**(5), **init**(8), **logintool**(8)

## DIAGNOSTICS

*ttyxx*: **No such device or address.**

*ttyxx*: **No such file or directory.**

A terminal which is turned on in the **ttys** file cannot be opened, likely because the requisite lines are either not configured into the system, the associated device was not attached during boot-time system configuration, or the special file in **/dev** does not exist.

**NAME**

gpconfig – initialize the Graphics Processor

**SYOPNSIS**

**/usr/etc/gpconfig** *gpunit* [ [ **–b** ] [ **–f** ] *fbunit*... ]

**DESCRIPTION**

**gpconfig** binds **cgtwo** frame buffers to the GP, (Graphics Processor) and loads and starts the appropriate microcode in the GP. For example, the command line:

**/usr/etc/gpconfig gpone0 cgtwo0 cgtwo1**

will bind the frame buffer boards **cgtwo0** and **cgtwo1** to the Graphics Processor **gpone0.** The devices **/dev/gpone0a** and **/dev/gpone0b** will then refer to the combination of **gpone** and **cgtwo0** or **cgtwo1** respectively.

The same **cgtwo** frame buffer cannot be bound to more than one GP.

All **cgtwo** frame buffer boards bound to a GP must be configured to the same width and height.

The standard version of the file **/etc/rc.local** contains the following **gpconfig** command line:

**/usr/etc/gpconfig gpone0 -f -b cgtwo0**

This binds **gpone0** and **cgtwo0** as **gpone0a**, causes **gpone0a** to use the Graphics Buffer Board if it is present, and redirects **/dev/fb** to be **/dev/gpone0a**. If another configuration is desired, edit the command line in **/etc/rc.local** to do the appropriate thing.

It is inadvisable to run the **gpconfig** command while the GP is being used. Unpredictable results may occur. If it is necessary to change the frame buffer bindings to the GP (or to stop using the GP altogether), bring the system down gently, boot single user, edit the **gpconfig** line in the **/etc/rc.local** file, and bring the system back up multiuser.

**OPTIONS**

**–b** Configure the GP to use the Graphics Buffer as well. Currently only one GP-to-frame-buffer binding is allowed to use the graphics buffer at a time. Only the last **-b** option in the command line takes effect.

**–f** Redirect **/dev/fb** to the device formed by binding *gpunit* with **fbunit.** Only the last **–f** option in the command line takes effect.

**FILES**

**/dev/cgtwo[0-9]**
**/dev/fb**
**/dev/gpone[0-3][abcd]**
**/usr/lib/gp1cg2.1024.ucode**
**/usr/lib/gp1cg2.1152.ucode**
**/etc/rc.local**

**SEE ALSO**

**cgtwo**(4S), **gpone**(4S)

## NAME

grpck – check group database entries

## SYNOPSIS

**/usr/etc/grpck** [*filename*]

## DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command.

**grpck** checks that a file in **group**(5) does not contain any errors; it checks the **/etc/group** file by default.

## FILES

**/etc/group**

## DIAGNOSTICS

**Too many/few fields**
An entry in the group file does not have the proper number of fields.

**No group name**
The group name field of an entry is empty.

**Bad character(s) in group name**
The group name in an entry contains characters other than lower-case letters and digits.

**Invalid GID**
The group ID field in an entry is not numeric or is greater than 65535.

**Null login name**
A login name in the list of login names in an entry is null.

**Login name not found in password file**
A login name in the list of login names in an entry is not in the password file.

## SEE ALSO

**groups**(1), **group**(5), **passwd**(5)

NAME

halt – stop the processor

SYNOPSIS

**/usr/etc/halt** [ **–nqy** ]

DESCRIPTION

**halt** writes out any information pending to the disks and then stops the processor.

**halt** normally logs the system shutdown to the system log daemon, **syslogd**(8), and places a shutdown record in the login accounting file **/var/adm/wtmp**. These actions are inhibited if the **–n** or **–q** options are present.

OPTIONS

**–n** Prevent the *sync* before stopping.

**–q** Do a quick halt. No graceful shutdown is attempted.

**–y** Halt the system, even from a dialup terminal.

FILES

**/var/adm/wtmp** login accounting file

SEE ALSO

**reboot**(8), **shutdown**(8), **syslogd**(8)

## NAME

htable – convert DoD Internet format host table

## SYNOPSIS

**/usr/etc/htable** *filename*

## DESCRIPTION

**htable** converts a host table in the format specified by RFC 952 to the format used by the network library routines. Three files are created as a result of running **htable**: **hosts**, **networks**, and **gateways**. The **hosts** file is used by the **gethostent**(3N) routines in mapping host names to addresses. The **networks** file is used by the **getnetent**(3N) routines in mapping network names to numbers. The **gateways** file is used by the routing daemon in identifying "passive" Internet gateways; see **routed**(8C) for an explanation.

If any of the files **localhosts**, **localnetworks**, or **localgateways** are present in the current directory, the file's contents is prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

**htable** is best used in conjunction with the **gettable**(8C) program which retrieves the DoD Internet host table from a host.

## FILES

**localhosts**
**localnetworks**
**localgateways**

## SEE ALSO

**intro**(3N), **gethostent**(3N), **getnetent**(3N), **gettable**(8C), **routed**(8C)

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *DoD Internet Host Table Specification*, RFC 952, Network Information Center, SRI International, Menlo Park, Calif., October 1985.

## BUGS

Does not properly calculate the **gateways** file.

## NAME

icheck – file system storage consistency check

## SYNOPSIS

**/usr/etc/icheck** [ **–s** ] **–b** *numbers* ] [ *filesystem* ]

## DESCRIPTION

Note: **icheck** has been superceded for normal consistency checking by **fsck**(8).

**icheck** examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of **icheck** includes a report of

The total number of files and the numbers of regular, directory, block special and character special files.

The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.

The number of free blocks.

The number of blocks missing; that is, not in any file nor in the free list.

With the **–s** option **icheck** ignores the actual free list and reconstructs a new one by rewriting the superblock of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the superblock will not continue to be used. Notice also that the words in the superblock which indicate the size of the free list and of the i-list are believed. If the superblock has been curdled these words will have to be patched. The **–s** option suppresses the normal output reports.

Following the **–b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

**icheck** is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

## FILES

Default file systems vary with installation.

## SEE ALSO

**fs**(5), **clri**(8), **dcheck**(8), **fsck**(8), **ncheck**(8)

## DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) **icheck** announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and **icheck** considers it to contain 0.

**Bad freeblock**
means that a block number outside the available space was encountered in the free list.

*n* **dups in free**
means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

## BUGS

Since **icheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous superblocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

## NAME

ifconfig – configure network interface parameters

## SYOPNSIS

**/etc/ifconfig** *interface* [ *address_family* ] [ *address* [ *dest_address* ] ] [ *parameters* ] [ **netmask** *mask* ] [ **broadcast** *address* ] ] [ **metric** *n* ]

**/etc/ifconfig** *interface* [ *protocol_family* ]

## DESCRIPTION

**ifconfig** is used to assign an address to a network interface and/or to configure network interface parameters. **ifconfig** must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. Used without options, **ifconfig** displays the current configuration for a network interface. If a protocol family is specified, **ifconfig** will report only the details specific to that protocol family. Only the super-user may modify the configuration of a network interface.

The *interface* parameter is a string of the form "name unit", for example **ie0**.

Since an interface may receive transmissions in differing protocols, each of which may require separate naming schemes, the parameters and addresses are interpreted according to the rules of some address family, specified by the *address_family* parameter. The address family currently supported is **inet**. If no address family is specified, **inet** is assumed.

For the DARPA Internet family (**inet**), the address is either a host name present in the host name data base (see **hosts**(5)) or in the Yellow Pages map **hosts**, or a DARPA Internet address expressed in the Internet standard "dot notation". Typically, an Internet address specified in dot notation will onsist of your system's network number and the machine's unique host number. A typical Internet address is **192.9.200.44**, where **192.9.200** is the network number and **44** is the machine's host number.

If the *dest_address* parameter is supplied in addition to the *address* parameter, it specifies the address of the correspondent on the other end of a point to point link.

## OPTIONS

The following *parameters* may be set with **ifconfig**:

**up** Mark an interface "up". This may be used to enable an interface after an "ifconfig down." It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

**down** Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.

**trailers** (**inet** only) Enable the use of a "trailer" link level encapsulation when sending (default — **really?**). If a network interface supports trailer encapsulation, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. This feature is machine-dependent, and therefore not recommended. On networks that support the Address Resolution Protocol (see **arp**(4P); currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailer encapsulation when sending to this host. Similarly, trailer encapsulations will be used when sending to other hosts that have made such requests.

**–trailers** Disable the use of a "trailer" link level encapsulation.

**arp** Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.

**–arp** Disable the use of the Address Resolution Protocol.

**metric** *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol (**routed**(8c)). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

**netmask mask** (**inet** only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table **networks**(5). The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion. If a + (plus sign) is given for the netmask value, then the network number is looked up in the netmasks.byaddr map of the Yellow Pages (or in the **/etc/netmasks**) file if not running Yellow Pages.

**broadcast** *address*
(**inet** only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 0's.

EXAMPLE

If your workstation is not attached to an Ethernet, the **ie0** interface should be marked "down" as follows:

```
ifconfig ie0 down
```

FILES

**/etc/netmasks**

SEE ALSO

**intro**(3N), **netmasks**(5), **netstat**(8C), **rc**(8)

DIAGNOSTICS

Messages indicating the specified interface does not exist, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

## NAME

inetd – Internet services daemon

## SYNOPSIS

**/usr/etc/inetd** [ **–d** ] [ *configuration-file* ]

## DESCRIPTION

**inetd,** the Internet services daemon, is normally run at boot time by the **/etc/rc.local** script. When started **inetd** reads its configuration information from *configuration-file*, the default being **/etc/inetd.conf.** See **inetd.conf**(5) for more information on the format of this file. It listens for connections on the Internet addresses of the services that its configuration file specifies. When a connection is found, it invokes the server daemon specified by that configuration file for the service requested. Once a server is finished, **inetd** continues to listen on the socket (except in some cases which will be described below).

Rather than having several daemon processes with sparsely distributed requests each running concurrently, **inetd,** reduces the load on the system by invoking Internet servers only as they are needed.

**inetd** itself provides a number of simple TCP-based services. These include **echo, discard, chargen** (character generator), **daytime** (human readable time), and **time** (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). For details of these services, consult the appropriate RFC, as listed below, from the Network Information Center.

**inetd** rereads its configuration file whenever it receives a hangup signal, **SIGHUP.** New services can be actived, and existing services deleted or modified in between whenever the file is reread.

## SEE ALSO

**inetd.conf**(5), **comsat**(8C), **ftpd**(8C), **rexecd**(8C), **rlogind**(8C), **rshd**(8C), **telnetd**(8C), **tftpd**(8C)

Postel, Jon, "Echo Protocol," RFC 862, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, "Discard Protocol," RFC 863, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, "Character Generater Protocol," RFC 864, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, "Daytime Protocol," RFC 867, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

Postel, Jon, and Ken Harrenstien, "Time Protocol," RFC 868, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

## NAME

infocmp – compare or print out terminfo descriptions

## SYNOPSIS

**infocmp** [ **–cdnILCruvV1** ] [ **–sd** ] [ **–si** ] [ **–sl** ] [ **–sc** ] [ **–w** *width* ] [**–A** *directory* ] [**–B** *directory* ] [ *termname* ... ]

## DESCRIPTION

**infocmp** compares a binary **terminfo**(5V) entry with other terminfo entries, rewrites a **terminfo** description to take advantage of the **use=***field*, or prints out a **terminfo** description from the corresponding binary file in a variety of formats. It displays boolean fields first, then numeric fields, then string fields.

It can also convert a **terminfo** entry to a **termcap**(5) entry; the **–C** flag causes **infocmp** to perform this conversion. Some **termcap** variables are not supported by **terminfo**, but those that can be derived from *terminfo* variables are displayed. Not all **terminfo** capabilities are translated either; only those that are allowed in a **termcap** entry are normally displayed. Specifying the **–r** option eliminates this restriction, allowing all capabilities to be displayed in **termcap** form.

Because padding is collected at the beginning of a capability, not all capabilities are displayed. Since mandatory padding is not supported by **terminfo** and **termcap** strings are not as flexible, it is not always possible to convert a **terminfo** string capability into an equivalent working **termcap** capability. Also, a subsequent conversion of the **termcap** file back into **terminfo** format will not necessarily reproduce the original source; **infocmp** attempts to convert parameterized strings, and comments out those that it can not.

Some common **terminfo** parameter sequences, their **termcap** equivalents, and some terminal types which commonly have such sequences, are:

| Terminfo | Termcap | Representative Terminals |
|---|---|---|
| %p1%c | %. | adm |
| %p1%d | %d | hp, ANSI standard, vt100 |
| %p1%'x'%+%c | %+x | concept |
| %i | %i | ANSI standard, vt100 |
| %p1%?%'x'%>%t%p1%'y'%+%; | %>xy | concept |
| %p2 is printed before %p1 | %r | hp |

If no *termname* arguments are given, the environment variable **TERM** is used for all expected *termname* arguments.

## OPTIONS

### Default Options

If no options are specified and either zero or one *termname* is specified, the **–I** option is assumed to be in effect. If more than one *termname* is specified, the **–d** option is assumed.

### Comparison Options

**infocmp** compares the description of the first terminal *termname* with each of the descriptions for terminals listed in subsequent *termname* arguments. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **–1** for integer variables, and **NULL** for string variables.

**–c** Produce a list of capabilities common to both entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **–u** option is worth using.

**–d** Produce a list of capabilities that differ between descriptions.

**–n** Produce a list of capabilities in neither entry.

### Source Listing Options

The **–I**, **–L**, and **–C** options produce a source listing for each terminal named.

**–I** Use the **terminfo** names.

**–L** Use the long C variable name listed in **<term.h>**.

**–C** Display only those capabilities that have **termcap** equivalents, using the **termcap** names and displaying them in **termcap** form whenever possible.

The source produced by the **–C** option may be used directly as a **termcap** entry, but not all of the parameterized strings may be changed to the **termcap** format. All padding information for strings is collected together and placed at the beginning of the string where **termcap** expects it. Mandatory padding (padding information with a trailing '/') will become optional.

**–r** When using **–C**, display all capabilities, not just those capabilities that have **termcap** equivalents.

**–u** Produce a **terminfo** source description for the first named terminal which is relative to the descriptions given by the entries for all terminals named subsequently on the command line, by analyzing the differences between them, and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using **infocmp** will show what can be done to change one description to be relative to the other.

A capability is displayed with an at-sign (@) if it no longer exists in the first terminal, but one of the other terminal entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries has a different value for that capability.

The order of the other *termname* entries is significant. Since the terminfo compiler **tic**(8V) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results, depending on the order in which they are given. **infocmp** flags any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability after a **use=** entry that contains it, will cause the second specification to be ignored. Using **infocmp** to recreate a description can be a useful check to make sure that everything was specified correctly in the original.

Specifying superfluous **use=** slows down the comparison, but is not fatal; **infocmp** flags superfluous **use=** fields.

**Sorting Options**

**–sd** Sort fields in the order that they are stored in the **terminfo** database.

**–si** Sort fields by **terminfo** name.

**–sl** Sort fields by the long C variable name.

**–sc** Sort fields by the **termcap** name.

If no sorting option is given, fields are sorted alphabetically by the **terminfo** name within each type, except in the case of the **–C** or the **–L** options, which cause the sorting to be done by the **termcap** name or the long C variable name, respectively.

**Changing Databases**

The location of the compiled **terminfo** database is taken from the environment variable **TERMINFO**. If the variable is not defined, or if the terminal is not found in that location, the system **terminfo** database, usually in **/usr/share/lib/terminfo**, is used. The options **–A** and **–B** may be used to override this location. With these options, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people.

**–A** Set **TERMINFO** for the first *termname* argument.

**–B** Set **TERMINFO** for the remaining *termname* arguments.

**Other Options**

**–v** Print out tracing information on the standard error.

**–V** Print out the version of the program in use on the standard error and exit.

**–1** Print fields out one to a line. Otherwise, fields are printed several to a line to a maximum width of 60 characters.

**–w** *width*
Change the output to *width* characters.

FILES

**/usr/share/lib/terminfo/?/***
compiled terminal description database

**/usr/5include/term.h**

SEE ALSO

**curses**(3V), **termcap**(5), **terminfo**(5V), **tic**(8V)

DIAGNOSTICS

**malloc is out of space!**
There was not enough memory available to process all the terminal descriptions requested. Run **infocmp** in several smaller stages (with fewer *termname* arguments).

**use= order dependency found:**
A value specified in one relative terminal specification was different from that in another relative terminal specification.

**'use=*term*' did not add anything to the description.**
A relative terminal name did not contribute anything to the final description.

**must have at least two terminal names for a comparison to be done.**
The **–u**, **–d** and **–c** options require at least two terminal names.

NAME

init – process control initialization

SYNOPSIS

**/usr/etc/init** [ **–bs** ]

DESCRIPTION

**init** is invoked inside the operating system as the last step in the boot procedure. It normally runs the sequence of commands in the script **/etc/rc.boot** (see **rc**(8)) to check the file system. If passed the **–b** flag from the boot program, **init** skips this step. If the file system check succeeds or is skipped, **init** runs the commands in **/etc/rc** and **/etc/rc.local** to begin multiuser operation; otherwise it commences single-user operation by giving the super-user a shell on the console. It is possible to pass the **–s** parameter from the boot program to **init** so that single-user operation is commenced immediately.

Whenever a single-user shell is created, and the system is running as a secure system, the **init** program demands the super-user password. This is to prevent an ordinary user from invoking a single-user shell and thereby circumventing the system's security. Logging out (for instance, by entering an EOT) causes **init** to proceed with a multi-user boot. The super-user password is demanded whenever the system is running secure as determined by **issecure**(3), or the terminal is labeled "secure" in **/etc/ttytab**.

Whenever single-user operation is terminated (for instance by killing the single-user shell) **init** runs the scripts mentioned above.

In multi-user operation, **init**'s role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file **/etc/ttytab** and executes a command for each terminal specified in the file. This command will usually be **/usr/etc/getty**. **getty**(8) opens and initializes the terminal line, reads the user's name and invokes **login**(1) to log in the user and execute the shell.

Ultimately the shell will terminate because it received EOF, either explicitly, as a result of hanging up, or from the user logging out. The main path of **init**, which has been waiting for such an event, wakes up and removes the appropriate entry from the file **/etc/utmp**, which records current users. **init** then makes an entry in **/var/adm/wtmp**, which maintains a history of logins and logouts. The **/var/adm/wtmp** entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and the command for that terminal is reinvoked.

**init** catches the *hangup* signal (SIGHUP) and interprets it to mean that the file **/etc/ttytab** should be read again. The shell process on each line which used to be active in **/etc/ttytab** but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add terminal lines without rebooting the system by changing **/etc/ttytab** and sending a *hangup* signal to the **init** process: use '**kill –HUP 1**'.

**init** terminates multi-user operations and resumes single-user mode if sent a terminate (SIGTERM) signal: use '**kill –TERM 1**'. If there are processes outstanding which are deadlocked (due to hardware or software failure), **init** does not wait for them all to die (which might take forever), but times out after 30 seconds and prints a warning message.

**init** ceases to create new processes, and allows the system to slowly die away, when sent a terminal stop (SIGTSTP) signal: use '**kill –TSTP 1**'. A later hangup will resume full multi-user operations, or a terminate will initiate a single-user shell. This hook is used by **reboot**(8) and **halt**(8).

Whenever it reads **/etc/ttytab**, **init** will normally write out an old-style **/etc/ttys** file reflecting the contents of **/etc/ttytab**. This is required in order that programs built on earlier versions of SunOS that read the **/etc/ttys** file (for example, programs using the **ttyslot**(3) routine, such as **shelltool (1)**) may continue to run. If it is not required that such programs run, **/etc/ttys** may be made a link (hard or symbolic) to **/etc/ttytab** and **init** will not write to **/etc/ttys**.

**init**'s role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the **init** program cannot be located, the system will print an error message and panic.

DIAGNOSTICS

*command* **failing, sleeping.**

A process being started to service a line is exiting quickly each time it is started. This is often caused by a ringing or noisy terminal line. **init** will sleep for 30 seconds, then continue trying to start the process.

**WARNING: Something is hung (won't die); ps axl advised.**

A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

FILES

**/dev/console**
**/dev/tty***
**/etc/utmp**
**/var/adm/wtmp**
**/etc/ttytab**
**/etc/rc**
**/etc/rc.local**
**/etc/rc.boot**
**/usr/etc/getty**

SEE ALSO

**kill**(1), **login**(1), **sh**(1), **shelltool**(1), **issecure**(3), **ttyslot**(3), **ttytab**(5), **getty**(8), **halt**(8), **rc**(8), **reboot**(8), **shutdown**(8)

## NAME

installboot – install bootblocks in a disk partition

## SYNOPSIS

**/usr/mdec/installboot** [ **–vlt** ] *bootfile protobootblk bootdevice*

## DESCRIPTION

The **boot**(8S) program is loaded from disk by bootblock code which resides in the bootblock area of a disk partition. In order for the bootblock code to read the boot program (usually **/boot**) it is necessary for it to know the block numbers occupied by the boot program. Previous versions of the bootblock code could find **/boot** by interpreting the file system on the partition from which it was being booted, but this is no longer so.

**installboot** plugs the block numbers of the boot program into a table in the bootblock code, and writes the modified bootblock code onto the disk. Note carefully that **installboot** must be run every time the boot program is reinstalled, since in general, the block list of the boot program will change each time it is written.

*bootfile* is the name of the boot program, usually **/boot**. *protobootblk* is the name of the bootblock code into which the block numbers of the boot program are to be inserted. The file read in must have an **a.out**(5) header, but it will be written out to the device with the header removed. *bootdevice* is the name of the disk device onto which the bootblock code is to be installed.

You can see how **installboot** works by making the destination a regular file instead of a device, and examining the result with **od**(1V).

## OPTIONS

**–v** Verbose. Display detailed information about the size of the boot program, etc.

**–l** Print out the list of block numbers of the boot program.

**–t** Test. Display various internal test messages.

## EXAMPLE

To install the bootblocks onto the root partition on a Xylogics disk:

```
example% cd /usr/mdec
example% installboot –vlt /boot bootxy /dev/xy0a
```

For an SD disk, you would use **bootsd** and **/dev/sd0a**, respectively, in place of **bootxy** and **/dev/xy0a**.

## SEE ALSO

**od**(1V), **init**(8), **boot**(8S), **bootparamd**(8), **kadb**(8S), **ndbootd**(8C), **monitor**(8S), **rc**(8), **reboot**(8)

*System and Network Administration*

*Installing the SunOS*

## NAME

iostat – report I/O statistics

## SYNOPSIS

**iostat** [ *interval* [ *count* ] ]

## DESCRIPTION

**iostat** iteratively reports the number of characters read and written to terminals, and, for each disk, the number of kilobytes transferred per second, and the milliseconds per average seek. It also gives the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, each fiftieth of a second, the state of each disk is examined and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices approximate average seek times are calculated for each device.

The optional *interval* argument causes **iostat** to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

## FILES

**/dev/kmem**
**/vmunix**

## SEE ALSO

**vmstat**(8)

**NAME**

ipallocd – Ethernet-to-IP address allocator

**SYNOPSIS**

**/usr/etc/rpc.ipallocd**

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

*ipallocd* is a daemon that determines or temporarily allocates IP addresses within a network segment. It has complete knowledge of the hosts listed in the yellow pages, and, if the system is running the name server, of any hosts listed in internet domain tables automatically accessed on that host through the standard library *gethostbyaddr( )* call.

This protocol uses DES authentication (the Sun Secure RPC protocol) to restrict access to this function. The only clients privileged to allocate addresses are those whose net IDs are in the *networks* group. For machine IDs, the machine must be a YP server.

The daemon uses permanent entries in the */etc/ethers* and */etc/hosts* files when they exist and are usable. In other cases, such as when a system is new to the network, *ipallocd* will enter a temporary mapping in a local cache. Entries in the cache are removed when there have been no references to a given entry in a 24-hour period. This cache survives system crashes so that IP addresses will remain consistent.

The daemon also provides corresponding IP address to name mapping.

*ipallocd* refuses to allocate addresses on networks not listed in the *netrange* file, or for which no free address is available.

**FILES**

**/etc/ez/ipalloc.cache**
**/etc/ez/ipalloc.netrange**

**SEE ALSO**

**pnp**(3R), **ipalloc**(3R), **ipallocd**(8C), **pnpboot**(8C), **netconfig**(8C)

## NAME

kadb – adb-like kernel and standalone-program debugger

## SYNOPSIS

**> b kadb** [ **–d** ] [ *boot-flags* ]

## DESCRIPTION

**kadb** is an interactive debugger that is similar in operation to **adb**(1), and runs as a standalone program under the PROM monitor. You can use **kadb** to debug the kernel, or to debug any standalone program.

Unlike **adb, kadb** runs in the same supervisor virtual address space as the program being debugged — although it maintains a separate context. The debugger runs as a *coprocess* that cannot be killed (no '**:k**') or rerun (no '**:r**'). There is no signal control (no '**:i**', '**:t**', or '**$i**'), although the keyboard facilities (CTRL–C, CTRL–S, and CTRL–Q) are simulated.

While the kernel is running under **kadb**, the abort sequence (**L1-A** or **BREAK**) drops the system into **kadb** for debugging — as will a system panic. When running other standalone programs under **kadb**, the abort sequence will pass control to the PROM monitor. **kadb** is then invoked from the monitor by jumping to the starting address for **kadb** found in **/usr/include/debug/debug.h** (currently this can be done for both Sun-2 and Sun-3 system machines with the monitor command '**g fd00000**', and with the monitor command '**g fe005000**' for Sun386i systems). **kadb**'s user interface is similar to **adb**. Note: **kadb** prompts with

**kadb>**

Most **adb** commands function in **kadb** as expected. Typing an abort sequence in response to the prompt returns you to the PROM monitor, from which you can examine control spaces that are not accessible within **adb** or **kadb**. The PROM monitor command **c** will return control to **kadb**. As with "**adb –k**", **$p** works when debugging kernels (by actually mapping in new user pages). The verbs **?** and **/** are equivalent in **kadb** , since there is only one address space in use.

## OPTIONS

**kadb** is booted from the PROM monitor as a standalone program. If you omit the **–d** flag, **kadb** automatically loads and runs **vmunix** from the filesystem **kadb** was loaded from. The **kadb vmunix** variable can be patched to change the default program to be loaded.

**–d** Interactive startup. Prompts with

**kadb:**

for a file to be loaded. From here, you can enter a boot sequence line to load a standalone program. Boot flags entered in response to this prompt are included with those already set and passed to the program. If you type a RETURN only, **kadb** loads **vmunix** from the filesystem that **kadb** was loaded from.

*boot-flags*

You can specify boot flags as arguments when invoking **kadb.** Note: **kadb** always sets the **–d** (debug) boot flag, and passes it to the program being debugged.

## USAGE

Refer to **adb** in *Debugging Tools.*

### Kernel Macros

As with **adb**, kernel macros are supported. With **kadb**, however, the macros are compiled into the debugger itself, rather than being read in from the filesystem. The **kadb** command **$M** lists macros known to **kadb.**

### Setting Breakpoints

Self-relocating programs such as the SunOS kernel need to be relocated before breakpoints can be used. To set the first breakpoint for such a program, start it with '**:s**'; **kadb** is then entered after the program is relocated (when the system initializes its interrupt vectors). Thereafter, '**:s**' single-steps as with **adb**. Otherwise, use '**:c**' to start up the program.

**Sun386i System Commands**

The Sun386i system version of **kadb** has the following additional commands. Note, for the general syntax of **adb** commands, see **adb**(1).

| | |
|---|---|
| **:i** | Read a byte (with the INB instruction) in from the port at *address*. |
| **:o** | Send a byte (with the OUTB instruction) containing *count* out through the port at *address*. |
| **:p** | Like **:b** in **adb**(1), but sets a breakpoint using the hardware debug register instead of the breakpoint instruction. The advantage of using **:p** is that when setting breakpoints with the debug register it is not necessary to have write access to the breakpoint location. Four (4) breakpoints can be set with the hardware debug registers. |
| **$S** | Switch I/O from the console to the serial port or vice versa. |
| **[** | Like **:e** in **adb**(1), but requires only one keystroke and no RETURN character. |
| **]** | Like **:s** in **adb**(1), but requires only one keystroke and no RETURN character. |

**Automatic Rebooting with kadb**

You can set up your workstation to automatically reboot **kadb** by patching the *vmunix* variable in **/boot** with the string **kadb**. (Refer to **adb** in *Debugging Tools* for details on how to patch executables.)

## FILES

**/vmunix**
**/boot**
**/kadb**
**/usr/include/debug/debug.h**

## SEE ALSO

**adb**(1), **boot**(8S)

*Debugging Tools*
*Writing Device Drivers*

## BUGS

There is no floating-point support, except on Sun386i systems.

**kadb** cannot reliably single-step over instructions that change the status register.

When sharing the keyboard with the operating system the monitor's input routines can leave the keyboard in a confused state. If this should happen, disconnect the keyboard momentarily and then reconnect it. This forces the keyboard to reset as well as initiating an abort sequence.

Most of the bugs listed in **adb**(1) also apply to **kadb**.

**NAME**

keyenvoy – talk to keyserver

**SYNOPSIS**

**keyenvoy**

**DESCRIPTION**

**keyenvoy** is used by some RPC programs to talk to the key server, **keyserv**(8C). The key server will not talk to anything but a root process, and **keyenvoy** is a set-uid root process that acts as an intermediary between a user process that wishes to talk to the key server and the key server itself.

This program cannot be run interactively.

**SEE ALSO**

**keyserv**(8C)

## NAME

keyserv – server for storing public and private keys

## SYNOPSIS

**keyserv** [ **–n** ]

## DESCRIPTION

**keyserv** is a daemon that is used for storing the private encryption keys of each user logged into the system. These encryption keys are using for accessing secure network services such as secure NFS. When a user logs in to the system, the **login(1)** program uses the login password to decrypt the user's encryption key stored in the Yellow Pages, and then gives the decrypted key to the **keyserv** daemon to store away.

Normally, root's key is read from the file **/etc/.rootkey** when the daemon starts up. This is useful during power-fail reboots when no one is around to type a password, yet you still want the secure network services to operate normally.

## OPTIONS

**–n** Do not read root's key from **/etc/.rootkey**. Instead, prompt the user for the password to decrypt **root 's** key stored in the Yellow Pages and then store the decrypted key in **/etc/.rootkey** for future use. This option is useful if the **/etc/.rootkey** file ever gets out of date or corrupted.

## FILES

**/etc/.rootkey**

## SEE ALSO

**login(1)**, **publickey(5)**

NAME

kgmon – generate a dump of the operating system's profile buffers

SYNOPSIS

**/usr/etc/kgmon** [ **–bhpr** ] [ *filesystem* ] [ *memory* ]

DESCRIPTION

**kgmon** is a tool used when profiling the operating system. When no arguments are supplied, **kgmon** indicates the state of operating system profiling as running, off, or not configured (see **config**(8)). If the **–p** flag is specified, **kgmon** extracts profile data from the operating system and produces a **gmon.out** file suitable for later analysis by **gprof**(1).

OPTIONS

**–b** Resume the collection of profile data.

**–h** Stop the collection of profile data.

**–p** Dump the contents of the profile buffers into a **gmon.out** file.

**–r** Reset all the profile buffers. If the **–p** flag is also specified, the **gmon.out** file is generated before the buffers are reset.

If neither **–b** nor **–h** is specified, the state of profiling collection remains unchanged. For example, if the **–p** flag is specified and profile data is being collected, profiling is momentarily suspended, the operating system profile buffers are dumped, and profiling is immediately resumed.

FILES

| | |
|---|---|
| **/vmunix** | the default system |
| **/dev/kmem** | the default memory |
| **gmon.out** | |

SEE ALSO

**gprof**(1), **config**(8)

DIAGNOSTICS

Users with only read permission on **/dev/kmem** cannot change the state of profiling collection. They can get a **gmon.out** file with the warning that the data may be inconsistent if profiling is in progress.

**NAME**

ldconfig – link-editor configuration

**SYNOPSIS**

**/usr/etc/ldconfig** [ *directory* ... ]

**DESCRIPTION**

**ldconfig** is used to configure a performance-enhancing cache for the run-time link-editor, **ld.so**. It is run from **/etc/rc.local** and periodically via **cron** to avoid linking with stale libraries. It should be also be run manually when a new shared object (e.g., a shared library) is installed on the system.

When invoked with no arguments, a default set of directories are built into the cache – these are the directories searched by default by the link editors. Additional directories may be specified on the command line.

**FILES**

**/etc/ld.so.cache** holds the cached data.

**SEE ALSO**

**ld**(1)

NAME

link, unlink – exercise link and unlink system calls

SYNOPSIS

**/usr/etc/link** *filename1 filename2*

**/usr/etc/unlink** *filename*

DESCRIPTION

**link** and **unlink** perform their respective system calls on their arguments, abandoning all error checking.

SEE ALSO

**rm**(1), **link**(2), **unlink**(2)

WARNINGS

Only the super-user can unlink a directory, in which case the files it contains are lost. The files can, however, be recovered from the file system's **lost+found** directory after performing an **fsck**.

If you have write permission on the directory in which *filename* resides, **unlink** removes that file without warning, regardless of its ownership.

## NAME

lockd – network lock daemon

## SYNOPSIS

**/etc/rpc.lockd** [ **–t** *timeout* ] [ **–g** *graceperiod* ]

## DESCRIPTION

**lockd** processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. **lockd** forwards lock requests for remote data to the server site's lock daemon through the **RPC/XDR(3N)** package. **lockd** then requests the status monitor daemon, **statd(8C),** for monitor service. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all client site lockds to submit reclaim requests. Client site lockds, on the other hand, are notified by the statd of the server recovery and promptly resubmit previously granted lock requests. If a lockd fails to secure a previously granted lock at the server site, the **lockd** sends SIGLOST to a process.

## OPTIONS

**–t** *timeout*

Use *timeout* (**seconds**) as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.

**–g** *graceperiod*

Use *graceperiod* (**seconds**) as the grace period duration instead of the default value (45 seconds).

## SEE ALSO

**fcntl**(2V), **lockf**(3), **signal**(3), **statd**(8C)

**NAME**

logintool – graphic login interface

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

**logintool** is invoked by **getty**(8) to display a full screen window for logging in. It cannot be run from the shell. It is more attractive than the traditional '**login:**' prompt, and also provides help for the person without a username and information about the workstation.

**logintool** is noramlly invoked on the console by **getty**(8), and works only on a frame buffer.

If the "newlogin" policy in the "policies" YP map is set to "unrestricted," then **logintool** may create new user accounts in the Yellow Pages. The account resides on the local system if it is diskful, or on the system's boot server if the local system is diskless.

**FILES**

*/usr/share/lib/ez/login*

**SEE ALSO**

**getty**(8)

## NAME

lpc – line printer control program

## SYNOPSIS

**/usr/etc/lpc** [ *command* [ *parameter*... ] ]

## DESCRIPTION

**lpc** controls the operation of the printer, or of multiple printers, as described in the **/etc/printcap** database. **lpc** commands can be used to start or stop a printer, disable or enable a printer's spooling queue, rearrange the order of jobs in a queue, or display the status of each printer—along with its spooling queue and printer daemon.

With no arguments, **lpc** runs interactively, prompting with **lpc>**. If arguments are supplied, **lpc** interprets the first as a *command* to execute; each subsequent argument is taken as a *parameter* for that command. The standard input can be redirected so that **lpc** reads commands from a file.

## USAGE

### Commands

Commands may be abbreviated to an unambiguous substring. Note: the *printer* parameter is specified just by the name of the printer (as **lw**), not as you would specify it to **lpr**(1) or **lpq**(1) (not as **–Plw**).

**?** [*command*]...

**help** [*command*]...

Display a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

**abort** [**all** | [*printer*...]]

Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by **lpr**(1)) for the specified printers. The **abort** command can only be used by the super-user.

**clean** [**all** | [*printer*...]]

Remove all files with names beginning with **cf**, **tf**, or **df** from the specified printer queue(s) on the local machine. The **clean** command can only be used by the super-user.

**disable** [**all** | [*printer*...]]

Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by **lpr**(1). The **disable** command can only be used by the super-user.

**down** [**all** | [*printer*...]] [*message*]

Turn the specified printer queue off, disable printing and put *message* in the printer status file. The message doesn't need to be quoted, the remaining arguments are treated like **echo**(1V). This is normally used to take a printer down and let others know why (**lpq**(1) indicates that the printer is down, as does the **status** command).

**enable** [**all** | [*printer*...]]

Enable spooling on the local queue for the listed printers, so that **lpr**(1) can put new jobs in the spool queue. The **enable** command can only be used by the super-user.

**exit**

**quit** Exit from **lpc**.

**restart** [**all** | [*printer*...]]

Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. **lpq**(1) reports that there is no daemon present when this condition occurs. This command can be run by any user.

**start** [**all** | [*printer*...]]

Enable printing and start a spooling daemon for the listed printers. The **start** command can only be used by the super-user.

**status** [ **all** | [ *printer* ... ] ]
Display the status of daemons and queues on the local machine. This command can be run by any user.

**stop** [ **all** | [ *printer* ... ] ]
Stop a spooling daemon after the current job completes and disable printing. The **stop** command can only be used by the super-user.

**topq** *printer* [ *job#* ... ] [ *user* ... ]
Move the print job(s) specified by *job#* or those job(s) belonging to *user* to the top (head) of the printer queue. The **topq** command can only be used by the super-user.

**up** [ **all** | [ *printer* ... ] ] Enable everything and start a new printer daemon. Undoes the effects of **down**.

FILES

| | |
|---|---|
| **/etc/printcap** | printer description file |
| **/var/spool/*** | spool directories |
| **/var/spool/*/lock** | lock file for queue control |

SEE ALSO

**lpq**(1), **lpr**(1), **lprm**(1), **printcap**(5), **lpd**(8)

DIAGNOSTICS

**?Ambiguous command**
The abbreviation you typed matches more than one command.

**?Invalid command**
You typed a command or abbreviation that was not recognized.

**?Privileged command**
You used a command can be executed only by the super-user.

## NAME

lpd – printer daemon

## SYNOPSIS

**/usr/lib/lpd** [ **–l** ] [ **–L** *logfile* ] [ *port#* ]

## DESCRIPTION

**lpd** is the line printer daemon (spool area handler). It is normally invoked at boot time from the **rc**(8) script, making a single pass through the **printcap**(5) file to find out about the existing printers and printing any files left after a crash. It then accepts requests to print files in a queue, transfer files to a spooling area, display a queue's status, or remove jobs from a queue. In each case, it forks a child process for each request, and continues to listen for subsequent requests.

The Internet port number used to communicate with other processes is normally obtained with **getservent**(3N), but can be specified with the *port#* argument.

## OPTIONS

**–l** Log valid requests received from the network. This can be useful for debugging purposes.

**–L** *logfile*
Change the file used for writing error conditions to *logfile*. The default is to report a message using the **syslog**(3) facility.

## OPERATION

### Access Control

Access control is provided by two means. First, all requests must come from one of the machines listed in either the file **/etc/hosts.equiv** or **/etc/hosts.lpd**. Second, if the **rs** capability is specified in the **printcap** entry, **lpr**(1) requests are only be honored for users with accounts on the printer host.

### Lock File

The **lock** file in each spool directory is used to prevent multiple daemons from becoming active, and to store information about the daemon process for **lpr**(1), **lpq**(1), and **lprm**(1).

**lpd** uses **flock**(2) to provide exclusive access to the lock file and to prevent multiple daemons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of **lpd** for the programs **lpq**(1) and **lprm**(1).

### Control Files

After the daemon has successfully set the lock, it scans the directory for files beginning with **cf**. Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character that indicates what to do with the remainder of the line.

- **J** Job name to print on the burst page.
- **C** Classification line on the burst page.
- **L** Literal. This line contains identification information from the password file, and causes a burst page to be printed.
- **T** Title string for page headings printed by **pr**(1V).
- **H** Hostname of the machine where **lpr**(1) was invoked.
- **P** Person. Login name of the person who invoked **lpr**(1). This is used to verify ownership by **lprm**(1).
- **M** Send mail to the specified user when the current print job completes.
- **f** Formatted File, the name of a file to print that is already formatted.
- **l** Like **f**, but passes control characters along, and does not make page breaks.
- **p** Name of a file to print using **pr**(1V) as a filter.
- **t** Troff File. The file contains **troff**(1) output (cat phototypesetter commands).
- **n** Ditroff File. The file contains device independent troff output.
- **d** DVI File. The file contains TEX output (DVI format from Stanford).
- **g** Graph File. The file contains data produced by **plot**(3X).

| | |
|---|---|
| **c** | Cifplot File. The file contains data produced by *cifplot*. |
| **v** | The file contains a raster image. |
| **r** | The file contains text data with FORTRAN carriage control characters. |
| **1** | Troff Font R. The name of a font file to use instead of the default. |
| **2** | Troff Font I. The name of the font file to use instead of the default. |
| **3** | Troff Font B. The name of the font file to use instead of the default. |
| **4** | Troff Font S. The name of the font file to use instead of the default. |
| **W** | Width. Changes the page width (in characters) used by **pr**(1V) and the text filters. |
| **I** | Indent. Specify the number of characters by which to indent the output. |
| **U** | Unlink. The name of file to remove upon completion of printing. |
| **N** | Filename. The name of the file being printed, or a blank for the standard input (when **lpr**(1) is invoked in a pipeline). |

**Data Files**

If a file can not be opened, an error message is logged using the **LOG_LPR** facility of **syslog**(3). **lpd** will try up to 20 times to reopen a file it expects to be there, after which it proceeds to the next file or job.

**Minfree File**

The file *minfree* in each spool directory contains the number of disk blocks to leave free so that the line printer queue won't completely fill the disk. The *minfree* file can be edited with your favorite text editor.

## FILES

| | |
|---|---|
| **/etc/printcap** | printer description file |
| **/var/spool/*** | spool directories |
| **/var/spool/*/minfree** | minimum free space to leave |
| **/dev/lp*** | line printer devices |
| **/dev/printer** | socket for local requests |
| **/etc/hosts.equiv** | hosts allowed equivalent host access |
| **/etc/hosts.lpr** | hosts allowed printer access only |

## SEE ALSO

**lpr**(1), **lpq**(1), **lprm**(1), **printcap**(5), **lpc**(8), **pac**(8)

**NAME**

mailstats – print statistics collected by sendmail

**SYNOPSIS**

**/usr/etc/mailstats** [ *filename* ]

**DESCRIPTION**

**mailstats** prints out the statistics collected by the **sendmail** program on mailer usage. These statistics are collected if the file indicated by the **S** configuration option of **sendmail** exists. The **mailstats** program first prints the time that the statistics file was created and the last time it was modified. It will then print a table with one row for each mailer specified in the configuration file. The first column is the mailer number, followed by the symbolic name of the mailer. The next two columns refer to the number of messages received by *sendmail*, and the last two columns refer to messages sent by *sendmail*. The number of messages and their total size (in 1024 byte units) is given. No numbers are printed if no messages were sent (or received) for any mailer.

You might want to add an entry to **/var/spool/cron/crontab/root** to reinitialize the statistics file once a night. Copy **/dev/null** into the statistics file or otherwise truncate it to reset the counters.

**FILES**

| | |
|---|---|
| **/etc/sendmail.st** | default statistics file |
| **/etc/sendmail.cf** | sendmail configuration file |
| **/var/spool/cron/crontab/root** | |
| **/dev/null** | |

**SEE ALSO**

**sendmail**(8)

**BUGS**

Mailstats should read the configuration file instead of having a hard-wired table mapping mailer numbers to names.

NAME

makedbm – make a Yellow Pages dbm file

SYNOPSIS

**makedbm** [ **–b** ] [ **–i** *yp_input_file* ] [ **–o** *yp_output_name* ] [ **–d** *yp_domain_name* ] [ **–m** *yp_master_name* ] *infile outfile*

**makedbm** [ **–u** *dbmfilename* ]

DESCRIPTION

**makedbm** takes *infile* and converts it to a pair of files in **ndbm**(3) format, namely *outfile*.**pag** and *outfile*.**dir**. Each line of the input file is converted to a single **dbm** record. All characters up to the first TAB or SPACE form the key, and the rest of the line is the data. If a line ends with '\', then the data for that record is continued on to the next line. It is left for the clients of the Yellow Pages to interpret #; **makedbm** does not itself treat it as a comment character. *infile* can be '–', in which case the standard input is read.

**makedbm** is meant to be used in generating **dbm** files for the Yellow Pages, and it generates a special entry with the key *yp_last_modified*, which is the date of *infile* (or the current time, if *infile* is '–').

OPTIONS

**–b** Interdomain. Propagate a map to all servers using the interdomain name server **named**(8C).

**–i** Create a special entry with the key *yp_input_file*.

**–o** Create a special entry with the key *yp_output_name*.

**–d** Create a special entry with the key *yp_domain_name*.

**–m** Create a special entry with the key *yp_master_name*. If no master host name is specified, *yp_master_name* will be set to the local host name.

**–u** Undo a **dbm** file. That is, print out a **dbm** file one entry per line, with a single space separating keys from values.

EXAMPLE

It is easy to write shell scripts to convert standard files such as **/etc/passwd** to the key value form used by **makedbm**. For example,

```
#!/bin/awk -f
BEGIN { FS = ":"; OFS = "\t"; }
{ print $1, $0 }
```

takes the **/etc/passwd** file and converts it to a form that can be read by **makedbm** to make the Yellow Pages file **passwd.byname**. That is, the key is a username, and the value is the remaining line in the **/etc/passwd** file.

SEE ALSO

**yppasswd**(1), **ndbm**(3), **named**(8C)

NAME

makedev, MAKEDEV – make system special files

SYNOPSIS

**/dev/MAKEDEV** *device-name...*

DESCRIPTION

**MAKEDEV** is a shell script normally used to install special files. It resides in the **/dev** directory, as this is the normal location of special files. Arguments to **MAKEDEV** are usually of the form *device-name*? where *device-name* is one of the supported devices listed in section 4 of the manual and '*?*' is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

**std** Create the *standard* devices for the system; for example, **/dev/console, /dev/tty**.

**local** Create those devices specific to the local site. This request runs the shell file **/dev/MAKEDEV.local**. Site specific commands, such as those used to setup dialup lines as "ttyd?" should be included in this file.

Since all devices are created using **mknod**(8), this shell script is useful only to the super-user.

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use **sh -x MAKEDEV** in case of trouble.

SEE ALSO

**intro**(4), **config**(8), **mknod**(8)

NAME

makekey – generate encryption key

SYNOPSIS

**/usr/lib/makekey**

DESCRIPTION

**makekey** improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

**makekey** is intended for programs that perform encryption (for instance, **ed**(1) and **crypt**(1)). Usually makekey's input and output will be pipes.

SEE ALSO

**crypt**(1), **ed**(1)

**NAME**

mc68881version – print the MC68881 mask number and approximate clock rate

**SYNOPSIS**

**/usr/etc/mc68881version**

**AVAILABILITY**

Sun-2, Sun-3, and Sun-4 systems only.

**DESCRIPTION**

**mc68881version** determines whether an MC68881 or MC68882 floating-point coprocessor is available, and if so, determines its apparent mask number and approximate clock rate and prints them on the standard output. The clock rate is derived by timing floating-point operations with **getrusage**(2) and is thus somewhat variable.

**SEE ALSO**

**getrusage**(2)

## NAME

mconnect – connect to SMTP mail server socket

## SYNOPSIS

**/usr/etc/mconnect** [ **–p** *port* ] [ **-r** ] [ *hostname* ]

## DESCRIPTION

**mconnect** opens a connection to the mail server on a given host, so that it can be tested independently of all other mail software. If no host is given, the connection is made to the local host. Servers expect to speak the Simple Mail Transfer Protocol (SMTP) on this connection. Exit by typing the **quit** command. Typing EOF will send an end of file to the server. An interrupt closes the connection immediately and exits.

## OPTIONS

**–p** *port* Specify the port number instead of the default SMTP port (number 25) as the next argument.

**–r** ''Raw'' mode: disable the default line buffering and input handling. This gives you a similar effect as **telnet** to port number 25, not very useful.

## FILES

**/usr/lib/sendmail.hf** help file for SMTP commands

## SEE ALSO

**sendmail**(8)

Postel, Jonathan B *Simple Mail Transfer Protocol*, RFC821 August 1982, SRI Network Information Center

**NAME**

mkfile – create a file

**SYNOPSIS**

**mkfile** [ **-nv** ] *size*[**k**|**b**|**m**] *filename* ...

**DESCRIPTION**

**mkfile** creates one or more files that are suitable for use as NFS-mounted swap areas. The sticky bit is set, and the file is padded with zeroes by default. The default *size* is in bytes, but it can be flagged as kilobytes, blocks, or megabytes, with the **k**, **b**, or **m** suffixes, respectively.

**OPTIONS**

**–n** Create an empty *filename*. The size is noted, but disk blocks aren't allocated until data is written to them.

**–v** Verbose. Report the names and sizes of created files.

## NAME

mkfs – construct a file system

## SYNOPSIS

**/usr/etc/mkfs** [ **–N** ] *special size* [ *nsect* ] [ *ntrack* ] [ *blksize* ] [ *fragsize* ] [ *ncpg* ] [ *minfree* ] [ *rps* ] [ *nbpi* ] [ *opt* ] [ *apc* ] [ *rot* ]

## DESCRIPTION

Note: file systems are normally created with the **newfs**(8) command.

**mkfs** constructs a file system by writing on the special file *special* unless the **–N** flag has been specified. The numeric *size* specifies the number of sectors in the file system. **mkfs** builds a file system with a root directory and a lost+found directory (see **fsck**(8)). The number of inodes is calculated as a function of the file system size. No boot program is initialized by **mkfs** (see **newfs**(8)).

## OPTIONS

The optional arguments allow fine tune control over the parameters of the file system.

*nsect* The number of sectors per track on the disk. The default is **32**.

*ntrack* The number of tracks per cylinder on the disk. The default is **16**.

*blksize* The primary block size for files on the file system. It must be a power of two, currently selected from **4096** or **8192** (the default).

*fragsize* The fragment size for files on the file system. The *fragsize* represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range **512** to **8192**. The default is **1024**.

*ncpg* The number of disk cylinders per cylinder group. This number must be in the range **1** to **32**. The default is **16**.

*minfree* The minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is **10%**.

*rps* The rotational speed of the disk, in revolutions per second. The default is **60**.

*nbpi* The number of bytes for which one inode block is allocated. This parameter is currently set at one inode block for every 2048 bytes.

*opt* Space or time optimization preference; **s** specifies optimization for space, **t** specifies optimization for time. The default is **t**.

*apc* The number of alternates per cylinder (SCSI devices only). The default is **0**.

*rot* The expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

## SEE ALSO

**dir**(5), **fs**(5), **fsck**(8), **newfs**(8), **tunefs**(8)

*System and Network Administration*
McKusick, Joy, Leffler; *A Fast File System for UNIX*,

## NOTES

**newfs**(8) is much to be preferred for most routine uses.

## NAME

mknod – build special file

## SYNOPSIS

**/usr/etc/mknod** *filename* [ **c** ] [ **b** ] *major minor*

**/usr/etc/mknod** *filename* **p**

## DESCRIPTION

**mknod** makes a special file. The first argument is the *filename* of the entry. In the first form, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (for example, unit, drive, or line number). Only the super-user is permitted to invoke this form of the **mknod** command.

In the second form, **mknod** makes a named pipe (FIFO).

The first form of **mknod** is only for use by system configuration people. Normally you should use /dev/MAKEDEV instead when making special files.

## SEE ALSO

**mknod**(2), **makedev**(8)

NAME

mkproto – construct a prototype file system

SYNOPSIS

**/usr/etc/mkproto** *special proto*

DESCRIPTION

**mkproto** is used to bootstrap a new file system. First a new file system is created using **newfs**(8). **mkproto** is then used to copy files from the old file system into the new file system according to the directions found in the prototype file **proto**. The prototype file contains tokens separated by SPACE or NEWLINE characters. The first tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **–bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or '–' to specify set-user-id mode or not. The third is **g** or '–' for the set-group-id mode. The rest of the mode is a three digit octal number giving the owner, group, and other read, write, execute permissions, see **chmod**(1V).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, **mkproto** makes the entries '.' and '..' and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
d—777 3 1
usr     d—777 3 1
        sh      ——755 3 1 /usr/bin/sh
        ken     d—755 6 1
                $
        b0      b—644 3 1 0 0
        c0      c—644 3 1 0 0
        $
$
```

SEE ALSO

**chmod**(1V), **fs**(5), **dir**(5), **fsck**(8), **newfs**(8)

BUGS

There should be some way to specify links.

There should be some way to specify bad blocks.

**mkproto** can only be run on virgin file systems. It should be possible to copy files into existent file systems.

NAME

modload – load a Sun386i module

SYNOPSIS

**modload** *filename* [ **–conf** *config_file* ] [ **–entry** *entry_point* ] [ **–exec** *exec_file* ] [ **–o** *output_file* ] [ **–nolink** ] [ **–A** *vmunix_file* ]

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**modload** loads a loadable module into a running system. The input file *filename* is an object file (**.o** file).

OPTIONS

**–conf** *config_file*

Use this configuration file to configure the loadable driver being loaded. The commands in this file are the same as those that the **config**(8) program recognizes. There are two additional commands, **blockmajor** and **charmajor**, shown in the configuration file example below.

**–entry** *entry_point*

This is the module entry point. This is passed by **modload** to **ld**(1) when the module is linked. The default module entry point name is '*xxx***init**'.

**–exec** *exec_file*

This is the name of a shell script or executable image file that will be executed if the module is successfully loaded. It is always passed the module id and module type as the first two arguments. For loadable drivers, the third and fourth arguments are the block major and character major numbers respectively. For a loadable system call, the third argument is the system call number.

**–o** *output_file*

This is the name of the output file that is produced by the linker. If this option is omitted, then the output file name is *filename*> without the '**.o**'.

**–nolink** This option can be used if **modload** has already been issued once and the output file already exists. One must take care that neither the kernel nor the module have changed.

**–A** *vmunix_file*

This is the file that is passed to the linker to resolve module references to kernel symbols. The default is **/vmunix**. The symbol file must be for the currently running kernel or the module is likely to crash the system.

EXAMPLE

```
controller      fdc0 at atmem csr 0x001000 irq 6 priority 3
controller      fdc2 at atmem csr 0x002000 irq 5 priority 2
disk            fd0 at fdc0 drive 0
disk            fd0 at fdc0 drive 1
disk            fd0 at fdc0 drive 2
device          fd0 at fdc2 drive 0 csr 0x003000 irq 4 priority 2
disk            fd0 at fdc2 drive 1
blockmajor 51
charmajor 52
```

SEE ALSO

**ld**(1), **modunload**(8), **modstat**(8)

**NAME**

modstat – display status of Sun386i modules

**SYNOPSIS**

**modstat** [ **–id** *module_id* ]

**AVAILABILITY**

Sun386i systems only.

**DESCRIPTION**

**modstat** displays the status of the loaded modules.

**OPTIONS**

**–id** *module_id*

Display status of only this module.

**SEE ALSO**

**modload**(8), **modunload**(8)

NAME

modunload – unload a Sun386i module

SYNOPSIS

**modunload** **–id** *module_id* [ **–exec exec_file** ]

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**modunload** unloads a loadable module from a running system. The *module_id* is the ID of the module as shown by **modstat**(8).

OPTIONS

**–exec** *exec_file*

This is the name of a shell script or executable image file that will be executed before the module is unloaded. It is always passed the module id and module type as the first two arguments. For loadable drivers, the third and fourth arguments are the block major and character major numbers respectively. For a loadable system call, the third argument is the system call number.

SEE ALSO

**modload**(8), **modstat**(8)

## NAME

monitor – system ROM monitor

## SYNOPSIS

**L1–A**

**BREAK**

## DESCRIPTION

The CPU board of the Sun workstation contains an EPROM (or set of EPROMs), called the *monitor*, that controls the system during startup. The monitor tests the system before attempting to boot the operating system. If you interrupt the boot procedure by holding down **L1** while typing **a** or **A** on the workstation keyboard (or **BREAK** if the console is a dumb terminal) the monitor issues the prompt:

```
>
```

and accepts commands interactively.

## USAGE

### Commands

**+|–** Increment or decrement the current address and display the contents of the new location.

**^C** *source destination n*
(CTRL-C) Copy, byte-by-byte a block of length *n* from the *source* address to the *destination* address.

**^I** *program* (CTRL-I) Display the compilation date and location of *program*.

**^T** *virtual_address*
(CTRL-T) Display the physical address to which *virtual_address* is mapped.

**a** [*n*] [*action*]... (Sun-2 and Sun-3 systems only)
Open **A**–register (cpu address register) *n*, and perform indicated actions. The number *n* can be any value from **0** to **7**, inclusive. The default value is **0**. A hexadecimal *action* argument assigns the value you supply to the register *n*. A non-hex *action* terminates command input.

**b** [*!*] [*device* [ (*c*,*u*,*p*) ] ] [*pathname* ] [*arguments_list* ]

**b[?]** Reset appropriate parts of the system and bootstrap a program. A '**!**' (preceding the *device* argument) prevents the system reset from occurring. Programs can be loaded from various devices (such as a disk, tape or Ethernet). '**b**' with no arguments will cause a default boot, either from a disk, or from an Ethernet controller. '**b?**' displays all boot devices and their *device* arguments, where *device* is one of:

| | |
|---|---|
| **ie** | Intel Ethernet |
| **le** | Lance Ethernet (Sun-2, Sun-3, Sun-4 systems only) |
| **sd** | SCSI disk |
| **st** | SCSI 1/4" tape |
| **mt** | Tape Master 9-track 1/2" tape (Sun-2, Sun-3, Sun-4 systems only) |
| **xd** | Xylogics 7053 disk (Sun-2, Sun-3, Sun-4 systems only) |
| **xt** | Xylogics 1/2" tape (Sun-2, Sun-3, Sun-4 systems only) |
| **xy** | Xylogics 440/450 disk (Sun-2, Sun-3, Sun-4 systems only) |
| **fd** | Diskette (Sun386i system only) |

*c* A controller number (**0** if only one controller),

*u* A unit number (**0** if only one driver), and

*p* A partition.

*pathname* A pathname for a program such as **/stand/diag**. **/vmunix** is the default.

*arguments_list*
A list of up to seven arguments to pass to the program being booted.

**c** [*virtual_address*]

Resume execution of a program. When given, *virtual_address* is the address at which execution will resume. The default is the current PC (EIP on Sun386i systems). Registers are restored to the values shown by the **a**, **d**, and **r** commands (for Sun-2 and Sun-3 systems), or by the **d** and **r** commands (for Sun-4 systems), or by the **d** command (for Sun386i systems).

**d** [*window_number*] (Sun-4 systems only)

Display (dump) the state of the processor. The processor state is observable only after:

- An unexpected trap was encountered.
- A user program dropped into the monitor (by calling *abortent*).
- The user manually entered the monitor by typing **L1–A** or **BREAK**.

The display consists of the following:

- The special registers: PSR, PC, nPC, TBR, WIM and Y
- Eight global registers, and
- 24 window registers (8 *in*, 8 *local*, and 8 *out*), corresponding to one of the 7 available windows. If a Floating-Point Unit is on board, its status register along with its 32 floating-point registers are also shown.

*window_number*

Display the indicated *window_number*, which can be any value between 0 and 6, inclusive. If no window is specified and the PSR's current window pointer contains a valid window number, registers from the window that was active just prior to entry into the monitor are displayed. Otherwise, registers from window 0 are displayed.

**d** (Sun386i systems only)

Display (dump) the state of the processor. This display consists of the registers, listed below:

| | |
|---|---|
| Processor Registers: | EAX, ECX, EDX, ESI, EDI, ESP, EBP, EFLAGS, EIP |
| Segment Registers: | ES, CS, SS, DS, FS, GS |
| Memory Management Registers: | GDTR, LDTR, IDTR, TR |
| Control Registers: | CR0, CR2, CR3 |
| Debug Registers: | DR0, DR1 , DR2 , DR3, DR6, DR7 |
| Test Registers: | TR6, TR7 |

The processor's state is observable only after an unexpected trap, a user program has "dropped" into the monitor (by calling monitor function abortentor) or the user has manually "broken " into the monitor (by typing **L1–A** on the Workstation console, or **BREAK** on the dumb terminal's keyboard.

**d** [*n*] [*action*]... (Sun-2 and Sun-3 systems only)

Open **D**–register (cpu data register) *n*, and perform indicated actions. The number *n* can be any value from **0** to **7**, inclusive. The default is **0**. See the **a** command for a description of *action*.

**e** [*virtual_address*] [*action*] ...

Open the 16 bit word at *virtual_address* (default zero). On Sun-2, Sun-3, and Sun-4 systems, the address is interpreted in the address space defined by the **s** command. See the **a** command for a description of *action*.

**f** *virtual_address1 virtual_address2 pattern* [*size* ] (Sun-3 and Sun-4 systems only)

Fill the bytes, words or long words from *virtual_address1* (lower) to *virtual_address2* (higher) with the constant, *pattern*. The *size* argument can take one of the following values

**b** byte format (the default)
**w** word format
**l** long word format

For example, the following command fills the address block from 0x1000 to 0x2000 with the word pattern, 0xABCD:

**f 1000 2000 ABCD W**

**g** [*vector* ] [*argument* ]
**g** [*virtual_address* ] [*argument* ]

Goto (jump to) a predetermined or default routine (first form), or to a user-specified routine (second form). The value of *argument* is passed to the routine. If the *vector* or *virtual_address* argument is omitted, the value in the PC is used as the address to jump to.

To set up a predetermined routine to jump to, a user program must, prior to executing the monitor's **g** command, set the variable ***romp->v_vector_cmd** to be equal to the virtual address of the desired routine. Predetermined routines need not necessarily return control to the monitor.

The default routine, defined by the monitor, prints the user-supplied *vector* according to the format supplied in *argument*. This format can be one of:

**%x** hexadecimal
**%d** decimal

**g0** (Sun-2, Sun-3, and Sun-4 only)

When the monitor is running as a result of the system being interrupted, force a panic and produce a crash dump.

**g4** When the monitor is running as a result of the system being interrupted, force a kernel stack trace.

**h** (Sun-3 and Sun-4 and Sun386i systems)

Display the help menu for monitor commands and their descriptions. To return to the monitor's basic command level, press ESCAPE or **q** before pressing RETURN.

**i** [*cache_data_offset*] [*action*] ... (Sun-3/200 series and Sun-4 systems only)

Modify cache data RAM command. Display and/or modify one or more of the
Modify cache data RAM command. Display and/or modify one or more of the cache data addresses. See the **a** command for a description of *action*.

**j** [*cache_tag_offset* ] [*action* ] ... (Sun-3/200 series and Sun-4 systems only)

Modify cache tag RAM command. Display and/or modify the contents of one or more of the cache tag addresses. See the **a** command for a description of *action*.

**k** [*reset_level* ]

Reset the system. If *reset_level* is:

**0** CPU reset only (Sun-2 and Sun-3 systems). Reset VMEbus, interrupt registers, video monitor (Sun-4 systems). This is the default. Reset video (Sun386i systems).
**1** Software reset.
**2** Power-on reset. Resets and clears the memory. Runs the EPROM-based diagnostic self test, which can take several minutes, depending upon how much memory is being tested.

**kb** Display the system banner.

**l** [*virtual_address*] [*action*]...
Open the long word (32 bit) at memory address *virtual_address* (default zero). On Sun-2, Sun-3 and Sun-4 systems, the address is interpreted in the address space defined by the **s** command (below). See the **a** command for a description of *action*.

**m** [*virtual_address*] [*action*]...
Open the segment map entry that maps *virtual_address* (default zero). On Sun-2, Sun-3 and Sun-4 systems, the address is interpreted in the address space defined by the **s** command. Not supported on Sun386i. See the **a** command for a description of *action*.

**nd** (Sun386i systems only)
**ne**
**ni** Disable, enable, or invalidate the cache, respectively

**o** [*virtual_address*] [action]...
Open the byte location specified by
*virtual_address* (default zero). On Sun-2, Sun-3 and Sun-4 systems, the address is interpreted in the address space defined by the **s** command. See the **a** command for a description of *action*.

**p** [*virtual_address*] [*action*]...
Open the page map entry that maps *virtual_address* (default zero) in the address space defined by the **s** command. See the **a** command for a description of *action*.

**p** [*port_address*] [[*nonhex_char* [*hex_value*] | *hex_value*] ...] (Sun386i systems only)
Display or modify the contents of one or more port I/O addresses in byte mode. Each port address is treated as a 8-bit unit. The optional *port_address*, argument, which is a 16-bit quantity, specifies the initial port I/O address. See the **e** command for argument descriptions.

**q** [*eeprom_offset*] [*action*]... (Sun-3 and Sun-4 systems only)
Open the EEPROM *eeprom_offset* (default zero) in the EEPROM address space. All addresses are referenced from the beginning or base of the EEPROM in physical address space, and a limit check is performed to insure that no address beyond the EEPROM physical space is accessed. On Sun386i systems, open the NVRAM *nvram_offset* (default zero). This command is used to display or modify configuration parameters, such as: the amount of memory to test during self test, whether to display a standard or custom banner, if a serial port (A or B) is to be the system console, etc. See the **a** command for a description of *action*.

**r** [*reg_name*] [[*nonhex_char* [*hex_value*] | *hex_value*] ...] (Sun386i systems only)
Display or modify one or more of the processor registers. If *reg_name* is specified (2 or 3 characters from the above list), that register is displayed first. The default is **EAX**. See note on register availability under the command **d** (for Sun386i systems). See the **e** command for argument descriptions.

**s** [*step_count*] (Sun386i systems only)
Single step the execution of the interrupted program. The *step_count* argument specifies the number of single steps to execute before displaying the monitor prompt. The default is 1.

**r** [*register_number*] [*action*]... (Sun-2 and Sun-3 systems only)
Display and/or modify the register indicated. *register_number* can be one of:

**CA** 68020 Cache Address Register
**CC** 68020 Cache Control Register
**CX** 68020 System and User Context
**DF** Destination Function code
**IS** 68020 Interrupt Stack Pointer
**MS** 68020 Master Stack Pointer
**PC** Program Counter
**SC** 68010 System Context

**SF** Source Function code
**SR** Status Register
**SS** 68010 Supervisor Stack Pointer
**UC** 68010 User Context
**US** User Stack Pointer
**VB** Vector Base

Alterations to these registers (except SC and UC) do not take effect until the next **c** command is executed. See the **a** command for a description of *action*.

**r** [*register_number* ] (Sun-4 systems only)
**r** [*register_type* ]
**r** [*w window_number* ]

Display and/or modify one or more of the IU or FPU registers.

A hexadecimal *register_number* can be one of:

**0x00—0x0f** window(0,i0)—window(0,i7), window(0,i0)—window(0,i7)
**0x16—0x1f** window(1,i0)—window(1,i7), window(1,i0)—window(1,i7)
**0x20—0x2f** window(2,i0)—window(2,i7), window(2,i0)—window(2,i7)
**0x30—0x3f** window(3,i0)—window(3,i7), window(3,i0)—window(3,i7)
**0x40—0x4f** window(4,i0)—window(4,i7), window(4,i0)—window(4,i7)
**0x50—0x5f** window(5,i0)—window(5,i7), window(5,i0)—window(5,i7)
**0x60—0x6f** window(6,i0)—window(6,i7), window(6,i0)—window(6,i7)
**0x70—0x77** g0, g1, g2, g3, g4, g5, g6, g7
**0x78—0x7d** PSR, PC, nPC, WIM, TBR, Y
**0x7e—0x9e** FSR, f0—f31

Register numbers can only be displayed after an unexpected trap, a user program has entered the monitor using the *abortent* function, or the user has entered the monitor by manually typing **L1–A** or **BREAK**.

If a *register_type* is given, the first register of the indicated type is displayed. *register_type* can be one of:

**f** floating-point
**g** global
**s** special

If **w** and a *window_number* (**0—6**) are given, the first *in*-register within the indicated window is displayed. If *window_number* is omitted, the window that was active just prior to entering the monitor is used. If the PSR's current window pointer is invalid, window 0 is used.

**s** [*code*] (Sun-2 and Sun-3 systems only)

Set or query the address space to be used by subsequent memory access commands. *code* is one of:

**0** undefined
**1** user data space
**2** user program space
**3** user control space
**4** undefined
**5** supervisor data space
**6** supervisor program space
**7** supervisor control space

If *code* is omitted, **s** displays the current address space.

**s** [*asi* ] (Sun-4 systems only)

Set or display the Address Space Identifier. With no argument, **s** displays the current Address Space Identifier. The *asi* value can be one of:

| | |
|---|---|
| **0x2** | control space |
| **0x3** | segment table |
| **0x4** | Page table |
| **0x8** | user instruction |
| **0x9** | supervisor instruction |
| **0xa** | user data |
| **0xb** | supervisor data |
| **0xc** | flush segment |
| **0xd** | flush page |
| **0xe** | flush context |
| **0xf** | cache data |

**t** [*program*] (Sun-3 systems only)

Trace the indicated standalone *program*. Works only with programs that do not affect interrupt vectors.

**u** [*echo*]
**u [ port ] [ options ] [**
**u** [*u*] [*virtual_address* ]

With no arguments, display the current I/O device characteristics including: current input device, current output device, BAUD rates for serial ports A and B, an input-to-output echo indicator, and virtual addresses of mapped UART devices. With arguments, set or configure the current I/O device. With the **u** argument (**uu**...), set the I/O device to be the *virtual_address* of a UART device currently mapped.

*echo* Can be either **e** to enable input to be echoed to the output device, or **ne**, to indicate that input is not echoed.

*port* Assign the indicated *port* to be the current I/O device. *port* can be one of:

- **a** serial port A
- **b** serial port B (except on Sun386i systems)
- **k** the workstation keyboard
- **s** the workstation screen

*baud_rate* Any legal BAUD rate.

*options* can be any combination of:

- **i** input
- **o** output
- **u** UART
- **e** echo input to output
- **ne** do not echo input
- **r** reset indicated serial port (**a** and **b** ports only)

If either **a** or **b** is supplied, and no *options* are given, the serial port is assigned for both input and output. If **k** is supplied with no options, it is assigned for input only. If **s** is supplied with no options, it is assigned for output only.

**v** *virtual_address1 virtual_address2* [*size*] (Sun-3 and Sun-4 systems only)

Display the contents of *virtual_address1* (lower) *virtual_address2* (higher) in the format specified by *size*:

**b** byte format (the default)
**w** word format
**l** long word format

Enter return to pause for viewing; enter another return character to resume the display. To terminate the display at any time, press the space bar.

For example, the following command displays the contents of virtual address space from address 0x1000 to 0x2000 in word format:

**v 1000 2000 W**

**w** [*virtual_address* ] [*argument* ] (Sun-3 and Sun-4 systems only)

Set the execution vector to a predetermined or default routine. Pass *virtual_address* and *argument* to that routine.

To set up a predetermined routine to jump to, a user program must, prior to executing the monitor's **w** command, set the variable ***romp->v_vector_cmd** to be equal to the virtual address of the desired routine. Predetermined routines need not necessarily return control to the monitor.

The default routine, defined by the monitor, prints the user-supplied *vector* according to the format supplied in *argument*. This format can be one of:

**%x** hexadecimal
**%d** decimal

**x** (Sun-3 and Sun-4 systems only)

Display a menu of extended tests. These diagnostics permit additional testing of such things as the I/O port connectors, video memory, workstation memory and keyboard, and boot device paths.

**y c** *context_number* (Sun-4 systems only)
**y p|s** *context_number virtual_address*

Flush the indicated context, context page, or context segment.

**c** flush context *context_number*
**p** flushe the page beginning at *virtual_address* within context *context_number*
**s** flush the segment beginning at *virtual_address* within context *context_number*

**z** [*number*] [*breakpoint_virtual_address* [*type*] [*len*]] (Sun386i systems only)

Set or reset breakpoints for debugging. With no arguments, this command displays the existing breakpoints. The *number* argument is a values from 0 to 3, corresponding to the processor debug registers, **DR0** to **DR3**, respectively. Up to 4 distinct breakpoints can be specified. If *number* is not specified then the monitor chooses a breakpoint number. The *breakpoint_virtual_address* argument specifies the breakpoint address. The *type* argument can be one of:

**x** Instruction Execution breakpoint (the default)
**m** for Data Write only breakpoint
**r** Data Reads and Writes only breakpoint.

The *len* argument can be one of: '**b**', '**w**', or '**l**', corresponding to the breakpoint field length of byte, word, or long-word, respectively. The default is '**b**'. Since the breakpoints are set in the on-chip registers, an instruction breakpoint can be placed in ROM code or in code shared by several tasks. If the *number* argument is specified but not *breakpoint_virtual_address*, the corresponding breakpoint is reset.

**z** [*virtual_address*] (Sun-3 systems only)
Set a breakpoint at *virtual_address* in the address space selected by the **s** command.

**FILES**

**/vmunix**

## NAME

mount, umount – mount and dismount filesystems

## SYNOPSIS

**/usr/etc/mount** [ **–p** ]
**/usr/etc/mount –a[fnv]** [ **–t** *type* ]
**/usr/etc/mount** [ **–fnrv** ] [ **–t** *type* ] [ **–o** *options* ] *filesystem directory*
**/usr/etc/mount** [ **–vfn** ] [ **–o** *options* ] *filesystem* | *directory*

**/usr/etc/umount** [ **–t** *type* ] [ **–h** *host* ]
**/usr/etc/umount –a[v]**
**/usr/etc/umount** [ **–v** ] *filesystem* | *directory* ...

## DESCRIPTION

**mount** attaches a named *filesystem* to the filesystem hierarchy at the pathname location *directory*, which must already exist. If *directory* has any contents prior to the **mount** operation, these remain hidden until the *filesystem* is once again unmounted. If *filesystem* is of the form *host*:*pathname*, it is assumed to be an NFS filesystem (type **nfs**).

**umount** unmounts a currently mounted filesystem, which can be specified either as a *directory* or a *filesystem*.

**mount** and **umount** maintain a table of mounted filesystems in **/etc/mtab**, described in **fstab**(5). If invoked without an argument, **mount** displays the contents of this table. If invoked with either a *filesystem* or *directory* only, mount searches the file **/etc/fstab** for a matching entry, and mounts the filesystem indicated in that entry on the indicated directory.

## MOUNT OPTIONS

**–p** Print the list of mounted filesystems in a format suitable for use in **/etc/fstab**.

**–a** All. Attempt to mount all the filesystems described in **/etc/fstab**. If a *type* argument is specified with **–t**, mount all filesystems of that type. Filesystems are not necessarily mounted in the order shown in **/etc/fstab**.

**–f** Fake an **/etc/mtab** entry, but do not actually mount any filesystems.

**–n** Mount the filesystem without making an entry in **/etc/mtab**.

**–v** Verbose. Display a message indicating each filesystem being mounted.

**–t** *type* Specify a filesystem type. The accepted types are **4.2**, and **nfs**; see **fstab**(5) for a description of these types.

**–r** Mount the specified filesystem read-only. This is a shorthand for:

**mount –o ro** *filesystem directory*

Physically write-protected and magnetic-tape filesystems must be mounted read-only. Otherwise errors occur when the system attempts to update access times, even if no write operation is attempted.

**–o** *options*

Specify filesystem *options* —list of comma-separated words from the list below. Some options are valid for all filesystem types, while others apply to a specific type only.

*options* valid on *all* filesystems:

| | |
|---|---|
| **rw**\|**ro** | Read/write or read-only. |
| **suid**\|**nosuid** | Setuid execution allowed or disallowed. |
| **grpid** | Create files with BSD semantics for the propagation of the group ID. Under this option, files inherit the GID of the directory in which they are created, regardless of the directory's set-GID bit. |

**noauto** Do not mount this filesystem that is currently mounted read-only. If the filesystem is not currently mounted, an error results.

The default is '**rw,suid**'.

*options* specific to **4.2** filesystems:

**quota|noquota** Usage limits are enforced, or are not enforced. The default is **noquota.**

*options* specific to **nfs** (NFS)filesystems:

**bg|fg** If the first attempt fails, retry in the background, or, in the foreground.
**retry=***n* The number of times to retry the mount operation.
**rsize=***n* Set the read buffer size to *n* bytes.
**wsize=***n* Set the write buffer size to *n* bytes.
**timeo=***n* Set the NFS timeout to *n* tenths of a second.
**retrans=***n* The number of NFS retransmissions.
**port=***n* The server IP port number.
**soft|hard** Return an error if the server does not respond, or continue the retry request until the server responds.
**intr** Allow keyboard interrupts on hard mounts.
**secure** Use a more secure protocol for NFS transactions.
**acregmin=***n* Hold cached attributes for at least *n* seconds after file modification.
**acregmax=***n* Hold cached attributes for no more than *n* seconds after file modification.
**acdirmin=***n* Hold cached attributes for at least *n* seconds after directory update.
**acdirmax=***n* Hold cached attributes for no more than *n* seconds after directory update.
**actimeo=***n* Set *min* and *max* times for regular files and directories to *n* seconds.

Regular defaults are:

**fg,retry=10000,timeo=7,retrans=3,port=NFS_PORT,hard,**
**acregmin=3,acregmax=60,acdirmin=30,acdirmax=60**

Defaults for **rsize** and **wsize** are set internally by the system kernel.

## UMOUNT OPTIONS

**-h** *host* Unmount all filesystems listed in **/etc/mtab** that are remote-mounted from *host*.

**-t** *type* Unmount all filesystems listed in **/etc/mtab** that are of a given *type*.

**-a** Unmount all filesystems currently mounted (as listed in **/etc/mtab**).

**-v** Verbose. Display a message indicating each filesystem being unmounted.

## NFS FILESYSTEMS

### Background vs. Foreground

Filesystems mounted with the **bg** option indicate that **mount** is to retry in the background if the server's mount daemon (**mountd**(8c)) does not respond. **mount** retries the request up to the count specified in the **retry=***n* option. Once the filesystem is mounted, each NFS request made in the kernel waits **timeo=***n* tenths of a second for a response. If no response arrives, the time-out is multiplied by 2 and the request is retransmitted. When the number of retransmissions has reached the number specified in the **retrans=***n* option, a filesystem mounted with the **soft** option returns an error on the request; one mounted with the **hard** option prints a warning message and continues to retry the request.

### Read-Write vs. Read-Only

Filesystems that are mounted **rw** (read-write) should use the **hard** option.

### Interrupting Processes With Pending NFS Requests

The **intr** option allows keyboard interrupts to kill a process that is hung while waiting for a response on a hard-mounted filesystem.

**Secure Filesystems**

The **secure** option must be given if the server requires secure mounting for the filesystem.

**File Attributes**

The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, then the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories. Setting **actimeo=***n* extends flush time by *n* seconds for both regular files and directories.

## SYSTEM V COMPATIBILITY

**System V File-Creation Semantics**

Ordinarily, when a file is created its GID is set to the effective GID of the calling process. This behavior may be overridden on a per-directory basis, by setting the set-GID bit of the parent directory; in this case, the GID is set to the GID of the parent directory (see **open**(2V) and **mkdir**(2)). Files created on filesystems that are mounted with the **grpid** option will obey BSD semantics; that is, the GID is unconditionally inherited from that of the parent directory.

## EXAMPLES

| | |
|---|---|
| To mount a local disk: | **mount /dev/xy0g /usr** |
| To fake an entry for **nd** root: | **mount -ft 4.2 /dev/nd0 /** |
| To mount all 4.2 filesystems: | **mount -at 4.2** |
| To mount a remote filesystem: | **mount -t nfs serv:/usr/src /usr/src** |
| To mount a remote filesystem: | **mount serv:/usr/src /usr/src** |
| To hard mount a remote filesystem: | **mount -o hard serv:/usr/src /usr/src** |
| To save current mount state: | **mount -p > /etc/fstab** |

## FILES

| | |
|---|---|
| **/etc/mtab** | table of mounted filesystems |
| **/etc/fstab** | table of filesystems mounted at boot |

## SEE ALSO

**mkdir**(2), **mount**(2), **unmount**(2), **open**(2V), **fstab**(5), **mtab**(5), **mountd**(8C), **nfsd**(8)

## BUGS

Mounting filesystems full of garbage crashes the system.

If the directory on which a filesystem is to be mounted is a symbolic link, the filesystem is mounted on *the directory to which the symbolic link refers*, rather than being mounted on top of the symbolic link itself.

## NAME

mountd – NFS mount request server

## SYNOPSIS

**/usr/etc/rpc.mountd** [ **–n** ]

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**mountd** is an RPC server that answers file system mount requests. It reads the file **/etc/xtab**, described in **exports**(5), to determine which file systems are available for mounting by which machines. It also provides information as to what file systems are mounted by which clients. This information can be printed using the **showmount**(8) command.

The **mountd** daemon is normally invoked by **rc**(8).

## OPTIONS

**–n** Do not check that the clients are root users. Though this option makes things slightly less secure, it does allow older versions (pre-3.0) of client NFS to work.

## FILES

**/etc/xtab**

## SEE ALSO

**exports**(5), **rc**(8), **showmount**(8)

**NAME**

named – Internet domain name server

**SYNOPSIS**

**/usr/etc/in.named** [ **–d** *level* ] [ **–p** *port* ] [ [ **–b** ] *bootfile* ]

**DESCRIPTION**

*filenamed* is the Internet domain name server. It is used by hosts on the DARPA Internet to provide access to the Internet distributed naming database. See RFC 1034 and RFC 1035 for more details. With no arguments *filenamed* reads **/etc/named.boot** for any initial data, and listens for queries on a privileged port.

**OPTIONS**

**–d** *level* Print debugging information. *level* is a number indicating the level of messages printed.

**–p** *port* Use a different *port* number.

**–b** *bootfile*
Use *bootfile* rather than **/etc/named.boot**.

**EXAMPLE**

```
;
;       boot file for name server
;
; type          domain          source file or host
;
domain          berkeley.edu
primary         berkeley.edu    named.db
secondary       cc.berkeley.edu 10.2.0.78 128.32.0.10
cache                   named.ca
```

The **domain** line specifies that **berkeley.edu** is the domain of the given server.

The **primary** line states that the file **named.db** contains authoritative data for **berkeley.edu**. The file **named.db** contains data in the master file format, described in RFC 1035, except that all domain names are relative to the origin; in this case, **berkeley.edu** (see below for a more detailed description).

The **secondary** line specifies that all authoritative data under **cc.berkeley.edu** is to be transferred from the name server at **10.2.0.78**. If the transfer fails it will try **128.32.0.10**, and continue for up to 10 tries at that address. The secondary copy is also authoritative for the domain.

The **cache** line specifies that data in **named.ca** is to be placed in the cache (typically such data as the locations of root domain servers). The file **named.ca** is in the same format as **named.db**.

The master file consists of entries of the form:

**$INCLUDE** <*filename*>
**$ORIGIN** <*domain*>
<*domain*> <*opt_ttl*> <*opt_class*> <*type*> <*resource_record_data*>

where *domain* is '.' for the root, '@' for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with '.', the current origin is appended to the domain. Domain names ending with '.' are unmodified.

The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero.

The *opt_class* field is currently one token, 'IN' for the Internet.

The *type* field is one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

**A** A host address (dotted quad).

**NS** An authoritative name server (domain).

**MX** A mail exchanger (domain).

**CNAME**
The canonical name for an alias (domain).

**SOA** Marks the start of a zone of authority (5 numbers). (see RFC 1035)).

**MB** A mailbox domain name (domain).

**MG** A mail group member (domain).

**MR** A mail rename domain name (domain).

**NULL** A null resource record (no format or data).

**WKS** A well know service description (not implemented yet).

**PTR** A domain name pointer (domain).

**HINFO** Host information (cpu_type OS_type).

**MINFO** Mailbox or mail list information (request_domain error_domain).

## FILES

| | |
|---|---|
| **/etc/named.boot** | name server configuration boot file |
| **/etc/named.pid** | the process ID |
| **/var/tmp/named.run** | debug output |
| **/var/tmp/named_dump.db** | dump of the name servers database |

## SEE ALSO

**kill**(1), **signal**(3), **resolver**(3), **resolve.conf**(5)

Mockapetris, Paul, *Domain Names - Concepts and Facilities*, RFC 1034, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain Names - Implementation and Specification*, RFC 1035, Network Information Center, SRI International, Menlo Park, Calif., November 1987.

Mockapetris, Paul, *Domain System Changes and Observations*, RFC 973, Network Information Center, SRI International, Menlo Park, Calif., January 1986.

Partridge, Craig, *Mail Routing and the Domain System*, RFC 974, Network Information Center, SRI International, Menlo Park, Calif., January 1986.

## NOTES

The following signals have the specified effect when sent to the server process using the **kill(1)** command.

**SIGHUP** Causes server to read named.boot and reload database.

**SIGQUIT**
Dumps current data base and cache to **/var/tmp/named_dump.db**.

**SIGEMT** Turns on debugging; each subsequent **SIGEMT** increments debug level.

**SIGFPE** Turns off debugging completely.

## NAME

ncheck – generate names from i-numbers

## SYNOPSIS

**/usr/etc/ncheck** [ **–i** *numbers* ] [ **–as** ] [ *filesystem* ]

## DESCRIPTION

Note: For most normal file system maintenance, the function of **ncheck** is subsumed by **fsck**(8).

**ncheck** with no argument generates a pathname versus i-number list of all files on a set of default file systems. Names of directory files are followed by '.'

A file system may be specified by the optional *filesystem* argument.

The report is in no useful order, and probably should be sorted.

## OPTIONS

**–i** *numbers*
: Report only those files whose i-*numbers* follow.

**–a**
: Print the names '.' and '..', which are ordinarily suppressed.

**–s**
: Report only special files and files with set-user-ID mode. This is intended to discover concealed violations of security policy.

## SEE ALSO

**sort**(1V), **dcheck**(8), **fsck**(8), **icheck**(8)

## DIAGNOSTICS

When the filesystem structure is improper, '**??**' denotes the "parent" of a parentless file and a pathname beginning with '...' denotes a loop.

## NAME

ndbootd – ND boot block server

## SYNOPSIS

**ndbootd** [ **–dv** ]

## DESCRIPTION

**ndbootd** sends boot blocks to diskless Sun-2 system clients that request them using the (now obsolete) ND protocol. This server uses the boot block contained in the file **/tftpboot/sun2.bb**. A client must appear in the **ethers**(5) and **hosts**(5) databases, in order for the request to be served. In determining whether to serve the client, **ndbootd** checks the **/tftpboot** directory for a file whose name is the client's IP address in hexadecimal notation. For example, if the file **/tftpboot/C00901AD** exists, the machine at IP address 192.9.1.173 can be served. This file normally contains the boot program that is sent to the client by **tftpd**(8C).

Only root can invoke **ndbootd**.

## OPTIONS

**–d** Debug. Display information about ignored packets, retransmissions, and address translation.

**–v** Verbose. Show a detailed listing of packets sent and received, etc.

If either option is used, all output is sent to the invoking terminal. Otherwise, error output (if any) appears on the console.

## FILES

| | |
|---|---|
| **/tftpboot** | bootfiles directory |
| **/tftpboot/sun2.bb** | boot blocks |
| **/tftpboot/????????** | boot programs for clients |

## SEE ALSO

**ethers**(5), **hosts**(5), **boot**(8S), **tftpd**(8C)

## NAME

netconfig – PNP boot service

## SYNOPSIS

**/single/netconfig** [ *–e* ] [ *–n* ]

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

**netconfig** is used both for automatic installation of new diskful systems, and during routine booting of all systems. The sequence of actions taken by **netconfig** depends on which of these situations is in effect, but it always sets the hostname, domainname, time, timezone, and interface IP address. If the system is newly installed on the network, it does more, perhaps interrogating the user about system configuration.

**netconfig** is invoked with the –e option from the */etc/rc.boot* script.

Invoked without options, *netconfig* may perform PNP set up, including set up of files, passwords, and secure RPCs. Unless **–n** is specified, it writes **/etc/net.conf**, which is read later by **rc.boot**. This includes the **VERBOSE** flag, derived from NVRAM data, which controls the verbosity of the commands in **rc.boot**.

### Routine Booting

Boot servers use information stored locally in YP maps rather than acquiring it over the network, except that they get the time from the *timehost* system if it is up. The following describes the steps taken by boot clients: diskful clients, diskless clients, and network clients.

Boot clients first invoke *rarp* to acquire an IP address. This is followed by a **ICMP Netmask** request to obtain the IP subnetwork mask, and then a **PNP_WHOAMI** RPC to determine the system's name, YP domain, and time zone. Then the systems clock is set using the RFC 868 time service. If **PNP_WHOAMI** fails, a **PNP_SETUP** sequence is followed by set up of **/etc/passwd** and other files.

## OPTIONS

**–e** Check shell environment variables. This option is specified during routine boot. **HOSTNAME** and **DOMAINNAME** are used to determine if the system is a YP server using local YP maps. Otherwise, if **NETWORKED** is **YES**, **netconfig** probes the network for network configuration. **MUST_SETUP** requires writing **/etc/passwd** and other files for setup in restricted network environments.

**–n** Used in conjunction with '**–e**', this does not probe the network for anything but just sets the hostname and domainname of the system from the environment variables **HOSTNAME** and **DOMAINNAME** respectively. Does not write the **/etc/net.conf** file.

## FILES

**/var/yp/***domainname***/netmasks**
**/var/yp/***domainname***/hosts**

## SEE ALSO

**pnpboot**(8C), **pnpd**(8C), **rarpd**(8C) **pnp**(3R)

## NAME

netstat – show network status

## SYNOPSIS

**netstat** [ **–aAn** ] [ **–f** *address_family* ] [ **system** ] [ **core** ]

**netstat** [ **–n** ] [ **–s** ] [ **–m** | **–i** | **–h** | **–r** ] [ **–f** *address_family* ] [ **system** ] [ **core** ]

**netstat** [ **–n** ] [ **–I** *interface* ] *interval* [ **system** ] [ **core** ]

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**netstat** displays the contents of various network-related data structures in various formats, depending on the options you select.

The first form of the command displays a list of active sockets for each protocol. The second form selects one from among various other network data structures. The third form displays running statistics of packet traffic on configured network interfaces; the *interval* argument indicates the number of seconds in which to gather statistics between displays.

The default value for the **system** argument is **/vmunix**; for *core*, the default is **/dev/kmem**.

## OPTIONS

**–a** Show the state of all sockets; normally sockets used by server processes are not shown.

**–A** Show the address of any protocol control blocks associated with sockets; used for debugging.

**–f** *address_family*
Limit statistics or address control block reports to those of the specified *address_family*, which can be one of:

**inet** For the AF_INET address family, or
**unix** For the AF_UNIX family.

**–h** Show the state of the IMP host table. (This does not work in an environment where the IMP host tables do not exist.)

**–i** Show the state of interfaces that have been auto-configured. Interfaces that are statically configured into a system, but not located at boot time, are not shown.

**–I** *interface*
Highlight information about the indicated *interface* in a separate column; the default (for the third form of the command) is the interface with the most traffic since the system was last rebooted. *interface* can be any valid interface listed in the system configuration file, such as **ie0** or **le0**.

**–m** Show the statistics recorded by management routines for the network's private buffer pool.

**–n** Show network addresses as numbers. **netstat** normally displays addresses as symbols. This option may be used with any of the display formats.

**–r** Show the routing tables. (When **–s** is also present, show routing statistics instead.)

**–s** Show per-protocol statistics. When used with the **–r** option, show routing statistics.

**–t** Replace queue length information with timer information.

## DISPLAYS

### Active Sockets (First Form)

The display for each active socket shows the local and remote address, the send and receive queue sizes (in bytes), the protocol, and the internal state of the protocol.

The symbolic format normally used to display socket addresses is either:

*hostname.port*

when the name of the host is specified, or:

*network.port*

if a socket address specifies a network but no specific host. Each **hostname** and *network* is shown according to its entry in the **/etc/hosts** or the **/etc/networks** file, as appropriate.

If the network or hostname for an address is not known (or if the **–n** option is specified), the numerical network address is shown. Unspecified, or "wildcard", addresses and ports appear as "*". (For more information regarding the Internet naming conventions, refer to **inet**(3N)).

*TCP Sockets*

The possible state values for TCP sockets are as follows:

| | |
|---|---|
| **CLOSED** | Closed: the socket is not being used. |
| **LISTEN** | Listening for incoming connections. |
| **SYN_SENT** | Actively trying to establish connection. |
| **SYN_RECEIVED** | Initial synchronization of the connection under way. |
| **ESTABLISHED** | Connection has been established. |
| **CLOSE_WAIT** | Remote shut down: waiting for the socket to close. |
| **FIN_WAIT_1** | Socket closed, shutting down connection. |
| **CLOSING** | Closed, then remote shutdown: awaiting acknowledgement. |
| **LAST_ACK** | Remote shut down, then closed: awaiting acknowledgement. |
| **FIN_WAIT_2** | Socket closed, waiting for shutdown from remote. |
| **TIME_WAIT** | Wait after close for remote shutdown retransmission. |

**Network Data Structures (Second Form)**

The form of the display depends upon which of the **–m**, **–i**, **–h** or **–r**, options you select. (If you specify more than one of these options, **netstat** selects one in the order listed here.)

*Routing Table Display*

The routing table display lists the available routes and the status of each. Each route consists of a destination host or network, and a gateway to use in forwarding packets. The *flags* column shows the status of the route (**U** if "up"), whether the route is to a gateway (**G**), and whether the route was created dynamically by a redirect (**D**).

Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface.

The **refcnt** column gives the current number of active uses per route. (Connection-oriented protocols normally hold on to a single route for the duration of a connection, whereas connectionless protocols obtain a route while sending to the same destination.)

The **use** column displays the number of packets sent per route.

The *interface* entry indicates the network interface utilized for the route.

**Cumulative Traffic Statistics (Third Form)**

When the *interval* argument is given, **netstat** displays a table of cumulative statistics regarding packets transferred, errors and collisions, the network addresses for the interface, and the maximum transmission unit ("mtu"). The first line of data displayed, and every 24th line thereafter, contains cumulative statistics from the time the system was last rebooted. Each subsequent line shows incremental statistics for the *interval* (specified on the command line) since the previous display.

SEE ALSO

**hosts**(5), **networks**(5), **protocols**(5), **services**(5), **iostat**(8), **trpt**(8C), **vmstat**(8)

**BUGS**

The notion of errors is ill-defined. Collisions mean something else for the IMP.

The kernel's tables can change while **netstat** is examining them, creating incorrect or partial displays.

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

**newaliases**

DESCRIPTION

**newaliases** rebuilds the random access data base for the mail aliases file **/etc/aliases**. It is run automatically by **sendmail**(8) (in the default configuration) whenever a message is sent.

FILES

**/etc/aliases**

SEE ALSO

**aliases**(5), **sendmail**(8)

## NAME

newfs – construct a new file system

## SYNOPSIS

**/usr/etc/newfs** [ **–nNv** ] [ *mkfs-options* ] *block-special-file*

## DESCRIPTION

**newfs** is a "friendly" front-end to the **mkfs**(8) program. On Sun systems, the disk type is determined by reading the disk label for the specified *block-special-file*.

*block-special-file* is the name of a block special device residing in **/dev**. If you want to make a file system on **sd0**, you can specify **sd0 rsd0** or **/dev/rsd0**; if you only specify **sd0**, **newfs** will find the proper device.

**newfs** then calculates the appropriate parameters to use in calling **mkfs**, builds the file system by forking **mkfs** and, if the file system is a root partition, installs the necessary bootstrap programs in its initial 16 sectors.

## OPTIONS

**–n** Do not install the bootstrap programs.

**–N** Print out the file system parameters without actually creating the file system.

**–v** Verbose. **newfs** prints out its actions, including the parameters passed to **mkfs**.

*mkfs-options*

Options that override the default parameters passed to **mkfs**(8) are:

**–b** *block-size*

The block size of the file system in bytes.

**–c** *#cylinders/group*

The number of cylinders per cylinder group in a file system. The default value used is 16.

**–d** *rotdelay*

This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

**–f** *frag-size*

The fragment size of the file system in bytes.

**–i** *bytes/inode*

This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

**–m** *free-space%*

The percentage of space reserved from normal users; the minimum free space threshold. The default value used is 10%.

**–o** *optimization*

(**space** or **time**). The file system can either be instructed to try to minimize the time spent allocating blocks, or to try to minimize the space fragmentation on the disk. If the minimum free space threshold (as specified by the **–m** option) is less than 10%, the default is to optimize for space; if the minimum free space threshold is greater than or equal to 10%, the default is to optimize for time.

**–r** *revolutions/minute*

The speed of the disk in revolutions per minute (normally 3600).

**–s** *size* The size of the file system in sectors.

**-t** *#tracks/cylinder*
The number of tracks per cylinders on the disk.

FILES

| | |
|---|---|
| **/usr/etc/mkfs** | to actually build the file system |
| **/usr/mdec** | for boot strapping programs /**dev** |

SEE ALSO

**fs**(5), **fsck**(8), **mkfs**(8), **tunefs**(8)

*System and Network Administration*

## NAME

newkey – create a new key in the publickey database

## SYNOPSIS

**newkey** [ **–h** *hostname* ] [ **–u** *username* ]

## DESCRIPTION

**newkey** is normally run by the network administrator on the YP master machine in order to establish public keys for users and super-users on the network. These keys are needed for using secure RPC or secure NFS.

**newkey** will prompt for the login password of the given username and then create a new public/secret key pair in **/etc/publickey** encrypted with the login password of the given user.

Use of this program is not required: users may create their own keys using **chkey (1).**

## OPTIONS

**–u** *username* Create a new public key for the given username. Prompts for the Yellow Pages password of the given username.

**–h** *hostname* Create a new public key for the super-user at the given hostname. Prompts for the root password of the given hostname.

## SEE ALSO

**keylogin(1), chkey(1), publickey(5), keyserv(8)**

## NAME

nfsd, biod – NFS daemons

## SYNOPSIS

**/usr/etc/nfsd** [*nservers*]

**/usr/etc/biod** [*nservers*]

## DESCRIPTION

**nfsd** starts the daemons that handle client filesystem requests. *nservers* is the number of file system request daemons to start. This number should be based on the load expected on this server. Four seems to be a good number.

**biod** starts *nservers* asynchronous block I/O daemons. This command is used on a NFS client to buffer cache handle read-ahead and write-behind. The magic number for *nservers* in here is also four.

When a file that is opened by a client is unlinked (by the server), a file with a name of the form **.nfs***XXX* (where *XXX* is a number) is created by the client. When the open file is closed, the **.nfs***XXX* file is removed. If the client crashes before the file can be closed, the **.nfs***XXX* file is not removed.

## FILES

| | |
|---|---|
| **.nfs***XXX* | client machine pointer to an open-but-unlinked file |

## SEE ALSO

**exports**(5), **mountd**(8C)

## NAME

nfsstat – Network File System statistics

## SYNOPSIS

**nfsstat** [ **–csnrz** ]

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**nfsstat** displays statistical information about the NFS (Network File System) and RPC (Remote Procedure Call), interfaces to the kernel. It can also be used to reinitialize this information. If no options are given the default is

**nfsstat –csnr**

That is, display everything, but reinitialize nothing.

## OPTIONS

**–c** Display client information. Only the client side NFS and RPC information will be printed. Can be combined with the **–n** and **–r** options to print client NFS or client RPC information only.

**–s** Display server information.

**–n** Display NFS information. NFS information for both the client and server side will be printed. Can be combined with the **–c** and **–s** options to print client or server NFS information only.

**–r** Display RPC information.

**–z** Zero (reinitialize) statistics. This option is for use by the super-user only, and can be combined with any of the above options to zero particular sets of statistics after printing them.

## DISPLAYS

The server RPC display includes the fields:

**calls** total number of RPC calls received

**badcalls** total number of calls rejected

**nullrecv** number of times no RPC packet was available when trying to receive

**badlen** number of packets that were too short

**xdrcall** number of packets that had a malformed header

The server NFS display shows the number of NFS calls received (**calls**) and rejected (**badcalls**), and the counts and percentages for the various calls that were made.

The client RPC display includes the following fields:

**calls** total number of RPC calls sent
**badcalls** total of calls rejected by a server
**retrans** number of times a call had to be retransmitted
**badxid** number of times a reply did not match the call
**timeout** number of times a call timed out
**wait** number of times a call had to wait on a busy CLIENT handle
**newcred** number of times authentication information had to be refreshed

The client NFS display shows the number of calls sent and rejected, as well as the number of times a CLIENT handle was received (**nclget**), the number of times a call had to sleep while awaiting a handle (**nclsleep**), as well as a count of the various calls and their respective percentages.

## FILES

**/vmunix** system namelist
**/dev/kmem** kernel memory

## NAME

nslookup – query name servers interactively

## SYNOPSIS

**nslookup** [ **–l** ] [ *address* ]

## DESCRIPTION

**nslookup** is an interactive program to query ARPA Internet domain name servers. The user can contact servers to request information about a specific host or print a list of hosts in the domain.

## OPTIONS

**–l** Use the local host's name server instead of the servers in **/etc/resolve.conf**. (If **/etc/resolve.conf** does not exist or does not contain server information, the **-l** option does not have any effect).

*address* Use the name server on the host machine with the given Internet address.

## USAGE

### Overview

The Internet domain name-space is tree-structured, with four top-level domains at present:

| | |
|---|---|
| **COM** | commercial establishments |
| **EDU** | educational institutions |
| **GOV** | government agencies |
| **MIL** | MILNET hosts |

If you are looking for a specific host, you need to know something about the host's organization in order to determine the top-level domain it belongs to. For instance, if you want to find the Internet address of a machine at UCLA, do the following:

1. Connect with the root server using the **root** command. The root server of the name space has knowledge of the top-level domains.
2. Since UCLA is a university, its domain name is **ucla.edu**. Connect with a server for the **ucla.edu** domain with the command **server**ucla.edu. The response will print the names of hosts that act as servers for that domain. Note: the root server does not have information about **ucla.edu**, but knows the names and addresses of hosts that do. Once located by the root server, all future queries will be sent to the UCLA name server.
3. To request information about a particular host in the domain (for instance, **locus**), just type the host name. To request a listing of hosts in the UCLA domain, use the **ls** command. The **ls** command requires a domain name (in this case, **ucla.edu**) as an argument.

Note: if you are connected with a name server that handles more than one domain, all lookups for host names must be fully specified with its domain. For instance, the domain **harvard.edu** is served by **seismo.css.gov**, which also services the **css.gov** and **cornell.edu** domains. A lookup request for the host **aiken** in the **harvard.edu** domain must be specified as **aiken.harvard.edu**. However, the

**set** *domain*=**name**

and

**set defname**

commands can be used to automatically append a domain name to each request.

After a successful lookup of a host, use the **finger** command to see who is on the system, or to finger a specific person. To get other information about the host, use the

**set** *querytype*=**value**

command to change the type of information desired and request another lookup. (**finger** requires the type to be A.)

**Commands**

Commands may be interrupted at any time by typing ^C. To exit, type ^D (EOF). The command line length must be less than 80 characters. Note: an unrecognized command will be interpreted as a host name.

*host* [*server*]
Look up information for *host* using the current default server or using *server* if it is specified.

**server** *domain*
**lserver** *domain*
Change the default server to *domain*. **lserver** uses the initial server to look up information about *domain* while **server** uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

**root** Changes the default server to the server for the root of the domain name space. Currently, the host **sri-nic.arpa** is used; this command is a synonym for '**lserver sri-nic.arpa**'.) The name of the root server can be changed with the **set root** command.

**finger** [ *name* ]
Connect with the finger server on the current host, which is defined by a previous successful lookup for a host's address information (see the **set** *querytype*=A command). As with the shell, output can be redirected to a named file using > and >>.

**"ls** [**–ah**]
List the information available for *domain*. The default output contains host names and their Internet addresses. The **-a** option lists aliases of hosts in the domain. The **-h** option lists CPU and operating system information for the domain. As with the shell, output can be redirected to a named file using > and >>. When output is directed to a file, hash marks are printed for every 50 records received from the server.

**view***filename*
Sort and list the output of the **ls** command with **more**(1).

**help**

**?** Print a brief summary of commands.

**set***keyword*[**=value**]
This command is used to change state information that affects the lookups. Valid keywords are:

**all** Prints the current values of the various options to **set**. Information about the current default server and host is also printed.

**[no]deb[ug]**
Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer. The default is **nodebug**.

**[no]def[name]**
Append the default domain name to every lookup. The default is **nodefname.**

**do[main]**=*filename*
Change the default domain name to *name*. The default domain name is appended to all lookup requests if **defname** option has been set. The default is the value in **/etc/resolve.conf.**

**q[querytype]**=*value*
Change the type of information returned from a query to one of:

A The host's Internet address (the default).
**CNAME**
The canonical name for an alias.
**HINFO** The host CPU and operating system type.
**MD** The mail destination.

**MX** The mail exchanger.
**MB** The mailbox domain name.
**MG** The mail group member.
**MINFO** The mailbox or mail list information.

(Other types specified in the RFC883 document are valid, but are not very useful.)

**[no]recurse**
Tell the name server to query other servers if it does not have the information. The default is **recurse**.

**ret[ry]=***count*
Set the number of times to retry a request before giving up to *count*. When a reply to a request is not received within a certain amount of time (changed with **set timeout**), the request is resent. The default is *count* is **2**.

**ro[ot]=***host*
Change the name of the root server to *host*. This affects the **root** command. The default root server is **sri-nic.arpa**.

**t[timeout]=***interval*
Change the time-out for a reply to *interval* seconds. The default *interval is* **10** seconds.

**[no]v[c]**
Always use a virtual circuit when sending requests to the server. The default is **novc**.

## DIAGNOSTICS

If the lookup request was not successful, an error message is printed. Possible errors are:

**Time-out**
The server did not respond to a request after a certain amount of time (changed with **set** *timeout*=**value**) and a certain number of retries (changed with **set** *retry*=**value**).

**No information**
Depending on the query type set with the **set querytype** command, no information about the host was available, though the host name is valid.

**Non-existent domain**
The host or domain name does not exist.

**Connection refused**
**Network is unreachable**
The connection to the name or finger server could not be made at the current time. This error commonly occurs with **finger** requests.

**Server failure**
The name server found an internal inconsistency in its database and could not return a valid answer.

**Refused**
The name server refused to service the request.

The following error should not occur and it indicates a bug in the program.

**Format error**
The name server found that the request packet was not in the proper format.

## FILES

**/etc/resolve.conf** initial domain name and name server addresses.

## SEE ALSO

**resolver**(3), **resolve.conf**(5), **named**(8C), RFC 882, RFC 883

## NAME

pac – printer/plotter accounting information

## SYNOPSIS

**/usr/etc/pac** [ **–cps** ] [ **–P***printer* ] [ **–p***price* ] [ *filename*... ]

## DESCRIPTION

**pac** reads the printer/plotter accounting files, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by e. ch user, and printing out how much each user consumed in pages or feet and dollars. If any *filenames* are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

## OPTIONS

**–c** Sorted the output by cost; usually the output is sorted alphabetically by name.

**–p***price* Use the value *price* for the cost in dollars instead of the default value of 0.02.

**–r** Reverse the sorting order.

**–s** Summarize the accounting information on the summary accounting file; this summary is necessary since on a busy system, the accounting file can grow by several lines per day.

**–P***printer*
Do accounting for the named *printer*. Normally, accounting is done for the default printer (site dependent) or the value of the **PRINTER** environment variable is used.

## FILES

| | |
|---|---|
| **/var/adm/?acct** | raw accounting files |
| **/var/adm/?_sum** | summary accounting files |

## BUGS

The relationship between the computed price and reality is as yet unknown.

## NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

## SYNOPSIS

**/usr/etc/ping** *host* [ *timeout* ]

**/usr/etc/ping** [ **–s** ] [ **–rv** ] *host* [ *packetsize* ] [ *count* ]

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**ping** utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from the specified *host* , or network gateway. ECHO_REQUEST datagrams, or "pings," have an IP and ICMP header, followed by a *struct* timeval, and then an arbitrary number of bytes to pad out the packet. If *host* responds, **ping** will print *host* **is alive** on the standard output and exit. Otherwise after *timeout* seconds, it will write **no answer from** *host*. The default value of *timeout* is 20 seconds.

When the **–s** flag is specified, **ping** sends one datagram per second, and prints one line of output for every ECHO_RESPONSE that it receives. No output is produced if there is no response. In this second form, **ping** computes round trip times and packet loss statistics; it displays a summary of this information upon termination or timeout. The default datagram packet size is 64 bytes, or you can specify a size with the *packetsize* command-line argument. If an optional *count* is given, **ping** sends only that number of requests.

When using **ping** for fault isolation, first '**ping**' the local host to verify that the local network interface is running.

## OPTIONS

**–r** Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to **ping** a local host through an interface that has been dropped by the router daemon, see **routed**(8C).

**–v** Verbose output. List any ICMP packets, other than ECHO_RESPONSE, that are received.

## SEE ALSO

**icmp**(4P), **ifconfig**(8C), **netstat**(8C), **rpcinfo**(8C), **spray**(8C)

NAME

pnpboot, pnp.s386 – pnp diskless boot service

SYNOPSIS — Sun386i

**/tftpboot/pnp.s386**

AVAILABILITY

Sun386i systems only.

DESCRIPTION

**pnp.s386** is a level 2 boot program that requests actions necessary to set up a diskless workstation on the network.

The PNP diskless boot service is used by diskless workstations at installation time to locate a server that will configure the diskless client.

The last steps of the level 1 boot (from the PROM) are to load the level 2 program through **rarpd**(8C) and **tftpd**(8C). The first step in the boot sequence is RARP to acquire an IP address. This is followed by TFTP service calls to acquire the **pnp.sun*** program file needed for the client's architecture. A PNP_ACQUIRE RPC is then broadcast to locate a server willing to configure the diskless client.

A PNP_SETUP is issued to the server which returns one of three statuses: success, failure, or in_progress. As long as the server responds with a status of in_progress the client will periodically issue a PNP_POLL until the status changes to either success or failure.

The last step is to reboot the client. This goes through a RARP, TFTP, BOOT sequence, with the boot using the normal **boot.sun*** file and **bootparamd**(8) service.

The system will have been set up using the IP address returned in the first step and a system name will have been assigned.

FILES

**/tftpboot/pnp.sun***

SEE ALSO

**bootparam**(3R), **bootparams**(5), **boot**(8S), **bootparamd**(8), **netconfig**(8C), **pnpd**(8C), **ipallocd**(8C), **rarpd**(8C), **tftpd**(8C)

## NAME

pnpd – PNP daemon

## SYNOPSIS

**/usr/etc/rpc.pnpd**

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

**pnpd** is used during routine booting of systems to determine their network configuration, and by new systems to configure themselves on a network. **pnpd** adds and removes diskless clients of the boot server on which it is running. The **pnpd** daemon is normally invoked in **rc.local**. The RPCs are used by **netconfig**(8C), **pnp.s386** (see **pnpboot**(8C)), and **client**(8).

The **bootservers** YP map specifies limits on server capacity and default swap size.

## FILES

**/export/exec/***arch*
: symbolic link to **/export/exec/***arch.release*

**/export/exec/***arch.release*
: symbolic link to **/usr** for the architecture

**/export/exec/***arch.release***/boot**
: root binaries

## SEE ALSO

**pnp**(3R), **client**(8), **ipallocd**(8C), **netconfig**(8C), **pnpboot**(8C)

## NAME

portmap – DARPA port to RPC program number mapper

## SYNOPSIS

**/usr/etc/rpc.portmap**

## DESCRIPTION

**portmap** is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell **portmap** what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact **portmap** on the server machine to determine the port number where RPC packets should be sent.

Normally, standard RPC servers are started by **inetd**(8C), so **portmap** must be started before **inetd** is invoked.

## SEE ALSO

**inetd.conf**(5), **rpcinfo**(8), **inetd**(8)

## BUGS

If **portmap** crashes, all servers must be restarted.

NAME

praudit – print contents of an audit trail file

SYNOPSIS

**praudit** [ **–lrs** ] [ **–d***del* ] [ *filename* ... ]

AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**praudit** reads the listed *filename*s (or standard input, if no *file*s are specified) and interprets the data as audit trail records as defined in **audit_control**(5). By default, times, security labels, user and group IDs are converted to their ASCII representation. Record type and event fields are converted to long ASCII representation.

OPTIONS

**–l** Print records one line per record. The record type and event fields are always converted to their short ASCII representation.

**–r** Print records in their raw form. Times, security labels, user IDs, group IDs, record types, and events are displayed as integers. Currently, in SunOS 4.0, labels are not used and are displayed as zero in this mode. This option and the **–s** option are exclusive. If both are used, a format usage error message is output.

**–s** Print records in their short form. All numeric fields are converted to ASCII and displayed. The short ASCII representations for the record type and event fields are used. Security labels are displayed in their short representation. Again, labels are not currently used. This option and the **–r** option are exclusive. If both are used, a format usage error message is output.

**–d***del* Use *del* as the field delimiter instead of the default delimiter, which is the comma. If *del* has special meaning for the shell, it must be quoted. The maximum size of a delimiter is four characters.

FILES

**/etc/passwd**

SEE ALSO

**audit**(2), **setuseraudit**(2), **getauditflags**(3), **audit_control**(5)

NAME

pstat – print system facts

SYNOPSIS

**/usr/etc/pstat** [ **–afipSsT** ] [ **–u** *pid* ] [ *system* [ *corefile* ] ]

DESCRIPTION

**pstat** interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in **/dev/kmem**. The required namelist is taken from **/vmunix** unless *system* is specified.

OPTIONS

**–a** Under **–p**, describe all process slots rather than just active ones.

**–f** Print the open file table with these headings:

LOC — The memory address of this table entry.

TYPE — The type of object the file table entry points to.

FLG — Miscellaneous state variables encoded thus:

- R open for reading
- W open for writing
- A open for appending
- S shared lock present
- X exclusive lock present
- I signal pgrp when data ready

CNT — Number of processes that know this open file.

MSG — Number of references from message queue.

DATA — The location of the vnode table entry or socket for this file.

OFFSET — The file offset (see **lseek**(2)).

**–i** Print the inode table including the associated vnode entries with these headings:

ILOC — The memory address of this table entry.

IFLAG — Miscellaneous inode state variables encoded thus:

- A inode access time must be corrected
- C inode change time must be corrected
- L inode is locked
- R inode is being referenced
- U update time (fs(5)) must be corrected
- W wanted by another process (L flag is on)

IDEVICE — Major and minor device number of file system in which this inode resides.

INO — I-number within the device.

MODE — Mode bits in octal, see **chmod**(2).

NLK — Number of links to this inode.

UID — User ID of owner.

SIZE/DEV — Number of bytes in an ordinary file, or major and minor device of special file.

VFLAG — Miscellaneous vnode state variables encoded thus:

- R root of its file system
- S shared lock applied
- E exclusive lock applied
- Z process is waiting for a shared or exclusive lock

CNT — Number of open file table entries for this vnode.

SHC — Reference count of shared locks on the vnode.

EXC — Reference count of exclusive locks on the vnode (this may be '> 1' if, for example, a file descriptor is inherited across a fork).

TYPE — Vnode file type, either VNON (no type), VREG (regular), VDIR (directory), VBLK (block device), VCHR (character device), VLNK (symbolic link), VSOCK (socket), VFIFO (named pipe), or VBAD (bad).

**-p** Print process table for active processes with these headings:

LOC — The memory address of this table entry.

S — Run state encoded thus:

0 no process
1 awaiting an event
2 (abandoned state)
3 runnable
4 being created
5 being terminated
6 stopped (by signal or under trace)

F — Miscellaneous state variables, ORed together (hexadecimal):

| | |
|---|---|
| 0000001 | loaded |
| 0000002 | a system process (scheduler or page-out daemon) |
| 0000004 | locked for swap out |
| 0000008 | swapped out during process creation |
| 0000010 | process is being traced |
| 0000020 | tracing parent has been told that process is stopped |
| 0000040 | user settable lock in memory |
| 0000080 | in page-wait |
| 0000100 | prevented from swapping during **fork**(2) |
| 0000200 | will restore old mask after taking signal |
| 0000400 | exiting |
| 0000800 | doing physical I/O |
| 0001000 | process resulted from a **vfork**(2) which is not yet complete |
| 0002000 | another flag for **vfork**(2) |
| 0004000 | process has no virtual memory, as it is a parent in the context of **vfork**(2) |
| 0008000 | process is demand paging pages from its executable image vnode |
| 0010000 | process has advised of sequential VM behavior with **vadvise**(2) |
| 0020000 | process has advised of random VM behavior with **vadvise**(2) |
| 0080000 | process is a session process group leader |
| 0100000 | process is tracing another process |
| 0200000 | process needs a profiling tick |
| 0400000 | process is scanning descriptors during select |
| 4000000 | process has done record locks |
| 8000000 | process is having its system calls traced |

PRI — Scheduling priority, see **getpriority**(2).

SIG — Signals received (signals 1-32 coded in bits 0-31),

UID — Real user ID.

SLP — Amount of time process has been blocked.

TIM — Time resident in seconds; times over 127 coded as 127.

CPU — Weighted integral of CPU time, for scheduler.

NI — Nice level, see **getpriority**(2).

PGRP — Process number of root of process group.

PID — The process ID number.

PPID — The process ID of parent process.

RSS — Resident set size — the number of physical page frames allocated to this process.

SRSS — RSS at last swap (0 if never swapped).

SIZE The size of the process image. That is, the sum of the data and stack segment sizes, not including the sizes of any shared libraries.
WCHAN Wait channel number of a waiting process.
LINK Link pointer in list of runnable processes.

**-S** Print the streams table with these headings:

LOC The memory address of this table entry.
WRQ The address of this stream's write queue.
VNODE The address of this stream's vnode.
DEVICE Major and minor device number of device to which this stream refers.
PGRP This stream's process group number.
FLG Miscellaneous stream state variables encoded thus:

- I waiting for **ioctl( )** to finish
- R read/recvmsg is blocked
- W write/putmsg is blocked
- P priority message is at stream head
- H device has been "hung up" (M_HANGUP)
- O waiting for open to finish
- M stream is linked under multiplexor
- D stream is in message-discard mode
- N stream is in message-nondiscard mode
- E fatal error has occurred (M_ERROR)
- T waiting for queue to drain when closing
- 2 waiting for previous **ioctl( )** to finish before starting new one
- 3 waiting for acknowledgment for **ioctl( )**
- B stream is in non-blocking mode
- A stream is in asynchronous mode
- o stream uses old-style no-delay mode
- S stream has had **TOSTOP** set
- C VTIME clock running
- V VTIME timer expired
- r collision on **select( )** for reading
- w collision on **select( )** for writing
- e collision on **select( )** for exceptional condition

The queues on the write and read sides of the stream are listed for each stream. Each queue is printed with these headings:

NAME The name of the module or driver for this queue.
COUNT The approximate number of bytes on this queue.
FLG Miscellaneous state variables encoded thus:

- E queue is enabled to run
- R someone wants to get from this queue when it becomes non-empty
- W someone wants to put on this queue when it drains
- F queue is full
- N queue should not be enabled automatically by a putq

MINPS The minimum packet size for this queue.
MAXPS The maximum packet size for this queue, or INF if there is no maximum.
HIWAT The high-water mark for this queue.
LOWAT The low-water mark for this queue.

**–s** Print information about swap space usage:

allocated: The amount of swap space (in bytes) allocated to private pages.

reserved: The number of swap space bytes not currently allocated, but claimed by memory mappings that have not yet created private pages.

used: The total amount of swap space, in bytes, that is either allocated or reserved.

available: The total swap space, in bytes, that is currently available for future reservation and allocation.

**–T** Print the number of used and free slots in the several system tables. This is useful for checking to see how full system tables have become if the system is under heavy load. Shows both used and cached inodes.

**–u** *pid* Print information about the process with ID *pid*.

FILES

| | |
|---|---|
| **/vmunix** | namelist |
| **/dev/kmem** | default source of tables |

SEE ALSO

**ps**(1), **chmod**(2), **fork**(2), **lseek**(2), **getpriority**(2), **stat**(2), **vadvise**(2), **vfork**(2), **fs**(5), **iostat**(8), **vmstat**(8)

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by **–T**; even more useful if these tables were dynamically allocated.

NAME

pwck – check password database entries

SYNOPSIS

**/usr/etc/pwck** [ *file* ]

DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command.

**pwck** checks that a file in **passwd**(5) does not contain any errors; it checks the **/etc/passwd** file by default.

FILES

**/etc/passwd**

DIAGNOSTICS

**Too many/few fields**
An entry in the password file does not have the proper number of fields.

**No login name**
The login name field of an entry is empty.

**Bad character(s) in login name**
The login name in an entry contains characters other than lower-case letters and digits.

**First char in login name not lower case alpha**
The login name in an entry does not begin with a lower-case letter.

**Login name too long**
The login name in an entry has more than 8 characters.

**Invalid UID**
The user ID field in an entry is not numeric or is greater than 65535.

**Invalid GID**
The group ID field in an entry is not numeric or is greater than 65535.

**No login directory**
The login directory field in an entry is empty.

**Login directory not found**
The login directory field in an entry refers to a directory that does not exist.

**Optional shell file not found.**
The login shell field in an entry refers to a program or shell script that does not exist.

**No netgroup name**
The entry is a Yellow Pages entry referring to a netgroup, but no netgroup is present.

**Bad character(s) in netgroup name**
The netgroup name in a Yellow Pages entry contains characters other than lower-case letters and digits.

**First char in netgroup name not lower case alpha**
The netgroup name in a Yellow pages entry does not begin with a lower-case letter.

SEE ALSO

**group**(5), **passwd**(5)

## NAME

pwdauthd – server for authenticating passwords

## SYNOPSIS

**/usr/etc/rpc.pwdauthd**

## AVAILABILITY

This program is available with the *Security* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**pwdauthd** is a server that determines authentication for users and groups. It handles authentication requests from **pwdauth**(3) and **grpauth**. Communication to and from **pwdauthd** is by means of RPC calls. The server is passed a *filename* and a *password*. It returns an integer value that specifies whether the *password* is valid. The possible return values are **PWA_VALID** if the name is valid, **PWA_INVALID** if the name is invalid, and **PWA_UNKNOWN** if validity cannot be determined because no adjunct files are present.

If **pwdauthd** is serving *pwdauth*, it determines whether the **passwd.adjunct** file exists. If not, it returns **PWA_UNKNOWN**. In this case, *pwdauth* knows to check the **/etc/passwd** file. Otherwise, the server calls **getpwanam** (see **getpwaent**(3)) to get the entry for *filename* in either the local or the Yellow Pages file for **passwd.adjunct**. If the encrypted password guess matches the encrypted password from the file, **pwdauthd** returns **PWA_VALID**. If the passwords do not match, it returns **PWA_INVALID**.

If **pwdauthd** is serving **grpauth**, it determines whether the **group.adjunct** file exists. If not, it returns **PWA_UNKNOWN**. In this case, **grpauth** knows to check the **/etc/group** file. Otherwise, the server calls **getgranam** (see **getgraent**(3)) to get the entry for *filename* in either the local or the Yellow Pages file for **group.adjunct**. If the encrypted password guess matches the encrypted password from the file, **pwdauthd** returns **PWA_VALID**. If the passwords do not match, it returns **PWA_INVALID**.

## SEE ALSO

**getpwaent**(3), **getgraent**(3), **pwdauth**(3)

**NAME**

quot – summarize file system ownership

**SYNOPSIS**

**/usr/etc/quot** [ **–acfhnv** ] [ *filesystem* ]

**DESCRIPTION**

**quot** displays the number of blocks (1024 bytes) in the named *filesystem* currently owned by each user.

**OPTIONS**

**–a** Generate are report for all mounted file systems.

**–c** Display three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

**–f** Display count of number of files as well as space owned by each user.

**–h** Estimate the number of blocks in the file — this doesn't account for files with holes in them.

**–n** Run the pipeline **ncheck filesystem | sort +0n | quot –n filesystem** to produce a list of all files and their owners.

**–v** Display three columns containing the number of blocks not accessed in the last 30, 60, and 90 days.

**FILES**

| | |
|---|---|
| **/etc/mtab** | mounted file systems |
| **/etc/passwd** | to get user names |

**SEE ALSO**

**du**(1V), **ls**(1V)

## NAME

quotacheck – file system quota consistency checker

## SYNOPSIS

**/usr/etc/quotacheck** [ **–v** ] [ **–p** ] *filesystem*...

**/usr/etc/quotacheck** [ **–apv** ]

## DESCRIPTION

**quotacheck** examines each file system, builds a table of current disk usage, and compares this table against that stored in the disk quota file for the file system. If any inconsistencies are detected, both the quota file and the current system copy of the incorrect quotas are updated (the latter only occurs if an active file system is checked).

**quotacheck** expects each file system to be checked to have a quota file named *quotas* in the root directory. If none is present, **quotacheck** will ignore the file system.

**quotacheck** is normally run at boot time from the **/etc/rc.local** file, see **rc**(8), before enabling disk quotas with **quotaon**(8).

**quotacheck** accesses the raw device in calculating the actual disk usage for each user. Thus, the file systems checked should be quiescent while **quotacheck** is running.

## OPTIONS

**–v** Indicate the calculated disk quotas for each user on a particular file system. **quotacheck** Normally reports only those quotas modified.

**–a** Check all the file systems indicated in **/etc/fstab** to be read-write with disk quotas.

**–p** Run parallel passes on the required file systems, using the pass numbers in **/etc/fstab** in an identical fashion to **fsck**(8).

## FILES

| | |
|---|---|
| **quotas** | quota file at the file system root |
| **/etc/mtab** | mounted file systems |
| **/etc/fstab** | default file systems |

## SEE ALSO

**quotactl**(2), **quotaon**(8), **rc**(8)

NAME

quotaon, quotaoff – turn file system quotas on and off

SYNOPSIS

**/usr/etc/quotaon** [ **–v** ] *filesystem...*

**/usr/etc/quotaon** [ **–av** ]

**/usr/etc/quotaoff** [ **–v** ] *filesystem...*

**/usr/etc/quotaoff** [ **–av** ]

DESCRIPTION OF QUOTAON

**quotaon** announces to the system that disk quotas should be enabled on one or more file systems. The file systems specified must be mounted at the time. The file system quota files must be present in the root directory of the specified file system and be named *quotas*.

DESCRIPTION OF QUOTAOFF

**quotaoff** announces to the system that file systems specified should have any disk quotas turned off.

OPTIONS TO QUOTAON

**–a** All file systems in **/etc/fstab** marked read-write with quotas will have their quotas turned on. This is normally used at boot time to enable quotas.

**–v** Display a message for each file system where quotas are turned on.

OPTIONS TO QUOTAOFF

**–a** Force all file systems in **/etc/fstab** to have their quotas disabled.

**–v** Display a message for each file system affected.

These commands update the status field of devices located in **/etc/mtab** to indicate when quotas are on or off for each file system.

FILES

| | |
|---|---|
| **quotas** | quota file at the file system root |
| **/etc/mtab** | mounted file systems |
| **/etc/fstab** | default file systems |

SEE ALSO

**quotactl**(2), **fstab**(5), **mtab**(5)

## NAME

rarpd – DARPA Reverse Address Resolution Protocol service

## SYNOPSIS

**/usr/etc/rarpd** *if hostname*

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rarpd** starts a daemon that responds to Reverse Address Resolution Protocol (Reverse ARP) requests. The daemon forks a copy of itself, and requires root privileges.

The Reverse ARP protocol is used by machines at boot time to discover their (32 bit) IP address given their (48 bit) Ethernet address. In order for the request to be answered, a machine's name-to-IP-address entry must exist in the **/etc/hosts** file and its name-to-Ethernet-address entry must exist in the **/etc/ethers** file. Furthermore, the server that runs the **rarpd** daemon must have entries in both files. Note that if the server machine is using the Yellow Pages service, the server's files are ignored, and the appropriate Yellow Pages maps queried.

The first argument, *if*, is one of the interface parameter strings (listed in **boot**(8S)), in the form of "name unit", for example **ie0**. The second argument, **hostname**, is the interface's corresponding host name. The *if*, *hostname* pair should be the same as the arguments passed to the **ifconfig (8)** command. As with **ifconfig, rarpd** must be invoked for each interface that the server wishes to support. Therefore a gateway machine may invoke the **rarpd** multiple times, for example:

**/usr/etc/rarpd ie0 host**
**/usr/etc/rarpd ie1 host-backbone**

## FILES

**/etc/ethers**
**/etc/hosts**

## SEE ALSO

**boot**(8S), **ifconfig**(8C) **ipallocd**(8C), **netconfig**(8C), **ethers**(5), **hosts**(5), **policies**(5), **netconfig**(8C), **ipallocd**(8C)

Finlayson, Ross, Timothy Mann, Jeffrey Mogul, and Marvin Theimer, *A Reverse Address Resolution Protocol*, RFC 903, Network Information Center, SRI International, Menlo Park, Calif., June 1984.

## NAME

rc, rc.boot, rc.local – command scripts for auto-reboot and daemons

## SYNOPSIS

**/etc/rc**

**/etc/rc.boot**

**/etc/rc.local**

## DESCRIPTION

**rc** and **rc.boot** are command scripts that are invoked by **init**(8) to perform file system housekeeping and to start system daemons. **rc.local** is a script for commands that are pertinent only to a specific site or client machine.

**rc.boot** sets the machine name, and then, if coming up multi-user, runs **fsck**(8) with the **–p** option. This "preens" the disks of minor inconsistencies resulting from the last system shutdown and checks for serious inconsistencies caused by hardware or software failure. If **fsck**(8) detects a serious disk problem, it returns an error and **init**(8) brings the system up in single-user mode. When coming up single-user, when **init**(8) is invoked by **fastboot**(8), or when it is passed the **–b** flag from **boot**(8S), functions performed in the **rc.local** file, including this disk check, are skipped.

Next, **rc** runs. If the system came up single-user, **rc** runs when the single-user shell terminates (see **init**(8)). It mounts 4.2 filesystems and spawns a shell for **/etc/rc.local**, which mounts NFS filesystems, and starts local daemons. After **rc.local** returns, **rc** starts standard daemons, preserves editor files, clears **/tmp**, starts system accounting (if applicable), starts the network (where applicable), and if enabled, runs **savecore**(8) to preserve the core image after a crash.

## Sun386i SYSTEM DESCRIPTION

These files operate as described above with the following variations:

**fsck**(8) is invoked with the **–y** option to prevent users being put in single-user mode by happenstance.

**rc.boot** invokes **netconfig**(8C) to configure the system for the network before booting. **netconfig** is invoked before the **/usr** filesystem is mounted, because **/usr** might be mounted from a server. **netconfig** writes **/etc/net.conf** unless the **–n** option is specified, controlling system booting.

The file **/etc/net.conf** stores these environment variables: The **VERBOSE** environment variable controls the verbosity of the messages from the **rc** script; its value is taken from NVRAM . The **NETWORKED** environment variable controls whether services useful only on a networked system are started in **/etc/rc.local**. The **PNP** environment variable, set up during initial system installation, controls whether local network configuration information is used or whether that information comes from the network. (Using automatic system installation causes all systems except boot servers to get this information from the network, facilitating network reconfiguration.) The **HOSTNAME** and **DOMAINNAME** environment variables, used together, help determine if this system is a boot server or, with **PNP** set to **no**, control the host name and domain name.

**rc.boot** dynamically loads device drivers.

**rc** invokes any programs found in **/var/recover** to clean up any operations partially completed when the system crashed or was shut down.

**rc.local** starts the automounter.

## FILES

**/etc/rc**
**/etc/rc.boot**
**/etc/rc.local**
**/etc/net.conf**
**/var/recover/***
**/var/yp/***
**/tmp**

SEE ALSO

**automount**(8), **boot**(8S), **fastboot**(8), **init**(8), **reboot**(8), **savecore**(8), **netconfig**(8C)

**NAME**

rdate – set system date from a remote host

**SYNOPSIS**

**/usr/ucb/rdate hostname**

**AVAILABILITY**

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**rdate** sets the local date and time from the **hostname** given as argument. You must be super-user on the local system. Typically **rdate** can be inserted as part of your **/etc/rc.local** startup script.

**FILES**

**/etc/rc.local**

**SEE ALSO**

**timed**(8C)

**BUGS**

Could be modified to accept a list of hostnames and try each until a valid date returned. Better yet would be to write a real date server that accepted broadcast requests.

## NAME

reboot – restart the operating system

## SYNOPSIS

**/usr/etc/reboot** [ **–dnq** ] [ *boot arguments* ]

## DESCRIPTION

**reboot** executes the **reboot**(2) system call to restart the kernel. The kernel is loaded into memory by the PROM monitor, which transfers control to it. See **boot**(8S) for details.

Although **reboot** can be run by the super-user at any time, **shutdown**(8) is normally used first to warn all users logged in of the impending loss of service. See **shutdown**(8) for details.

**reboot** performs a **sync**(1) operation on the disks, and then a multiuser reboot is initiated. See **init**(8) for details.

**reboot** normally logs the reboot to the system log daemon, **syslogd**(8), and places a shutdown record in the login accounting file **/var/adm/wtmp**. These actions are inhibited if the **–n** or **–q** options are present.

### Power Fail and Crash Recovery

Normally, the system will reboot itself at power-up or after crashes.

## OPTIONS

**–d** Dump system core before rebooting.

**–n** Avoid the **sync**(1). It can be used if a disk or the processor is on fire.

**–q** Quick. Reboots quickly and ungracefully, without first shutting down running processes.

### Boot Arguments

If a boot argument string is given, it is passed to the boot command in the PROM monitor. The string must be quoted if it contains spaces or other characters that could be interpreted by the shell. If the first character of the boot argument string is a minus sign '–' the string must be preceded by an option terminator string '– –' For example: 'reboot – – –s' to reboot and come up single user, '**reboot vmunix.test**' to reboot to a new kernel. See **boot**(8S) for details.

## FILES

**/var/adm/wtmp** login accounting file

## SEE ALSO

**sync**(1), **reboot**(2), **fsck**(8), **halt**(8), **init**(8), **shutdown**(8), **syslogd**(8), **boot**(8S), **crash**(8S)

## NAME

renice – alter priority of running processes

## SYNOPSIS

**/usr/etc/renice** *priority pid...*

**/usr/etc/renice** *priority* [ **–p** *pid...* ] [ **–g** *pgrp...* ] [ **–u** *username...* ]

## DESCRIPTION

**renice** alters the scheduling priority of one or more running processes.

## OPTIONS

By default, the processes to be affected are specified by their process IDs. *priority* is the new priority value.

**–p** *pid...*
Specify a list of process IDs.

**–g** *pgrp...*
Specify a list of process group IDs. The processes in the specified process groups have their scheduling priority altered.

**–u** *user...*
Specify a list of user IDs or usernames. All processes owned by each *user* have their scheduling altered.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to 20. (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range –20 — 20. Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to), 0 (the "base" scheduling priority) and any negative value (to make things go very fast).

If no **who** parameter is specified, the current process (alternatively, process group or user) is used.

## FILES

**/etc/passwd** to map user names to user ID's

## SEE ALSO

**pstat**(8)

## BUGS

If you make the priority very negative, then the process cannot be interrupted.

To regain control you must make the priority greater than zero.

Users other than the super-user cannot increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

## NAME

repquota – summarize quotas for a file system

## SYNOPSIS

**/usr/etc/repquota** [ **–v** ] *filesystem*...

**/usr/etc/repquota** [ **–av** ]

## DESCRIPTION

**repquota** prints a summary of the disc usage and quotas for the specified file systems. For each user the current number of files and amount of space (in kilobytes) is printed, along with any quotas created with **edquota**(8).

## OPTIONS

**–a** Report on all file systems indicated in **/etc/fstab** to be read-write with quotas.

**–v** Report all quotas, even if there is no usage.

Only the super-user may view quotas which are not their own.

## FILES

| | |
|---|---|
| **quotas** | quota file at the file system root |
| **/etc/fstab** | default file systems |

## SEE ALSO

**edquota**(8), **quota**(1), **quotacheck**(8), **quotactl**(2), **quotaon**(8)

## NAME

restore, rrestore – incremental file system restore

## SYNOPSIS

**/usr/etc/restore** *options* [ *filename*... ]

## DESCRIPTION

**restore** restores files from backup tapes created with the **dump**(8) command. *options* is a string of at least one of the options listed below, along with any modifiers and arguments you supply. Remaining arguments to **restore** are the names of files (or directories whose files) are to be restored to disk. Unless the **h** modifier is in effect, a directory name refers to the files it contains, and (recursively) its subdirectories and the files they contain.

## OPTIONS

**i** Interactive. After reading in the directory information from the tape, **restore** invokes an interactive interface that allows you to browse through the dump tape's directory hierarchy, and select individual files to be extracted. See **Interactive Commands**, below, for a description of available commands.

**r** Restore the entire tape. Load the tape's full contents into the current directory. This option should only be used to restore a complete dump tape onto a clear filesystem, or to restore an incremental dump tape after a full "level **0**" restore. For example:

```
example# /usr/etc/newfs /dev/rxy0g
example# /usr/etc/mount /dev/xy0g /mnt
example# cd /mnt
example# restore r
```

is a typical sequence to restore a "level **0**" dump. Another **restore** can be done to get an incremental dump in on top of this.

**R** Resume restoring. **restore** requests a particular tape of a multivolume set from which to resume a full restore (see the **r** option above). This allows **restore** to start from a checkpoint when it is interrupted in the middle of a full restore.

**t** Table of contents. List each *filename* that appears on the tape. If no *filename* argument is given, the root directory is listed. This results in a list of all files on the tape, unless the **h** modifier is in effect. (The **t** option replaces the function of the old **dumpdir** program).

**x** Extract the named files from the tape. If a named file matches a directory whose contents were written onto the tape, and the **h** modifier is not in effect, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no *filename* argument is given, the root directory is extracted. This results in the entire tape being extracted unless the **h** modifier is in effect.

### Modifiers

Some of the following modifiers take arguments that are given as separate words on the command line. When more than one such modifier appears within *options*, the arguments must appear in the same order as the modifiers that they apply to.

**c** Convert the contents of the dump tape to the new filesystem format.

**d** Debug. Turn on debugging output.

**h** Extract the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

**m** Extract by inode numbers rather than by filename to avoid regenerating complete pathnames. This is useful if only a few files are being extracted.

**v** Verbose. **restore** displays the name of each file it restores, preceded by its file type.

**y** Do not ask whether to abort the restore in the event of tape errors. **restore** tries to skip over the bad tape block(s) and continue as best it can.

**b** *factor*
Blocking factor. Specify the blocking factor for tape reads. By default, **restore** will attempt to figure out the block size of the tape. Note: a tape block is 512 bytes.

**f** *dump-file*
Use *dump-file* instead of **/dev/rmt?** as the file to restore from. If *dump-file* is specified as '–', **restore** reads from the standard input. This allows, **dump**(8) and **restore** to be used in a pipeline to dump and restore a file system:

**example# dump 0f – /dev/rxy0g |**

If the name of the file is of the form *machine*:*device* the restore is done from the specified machine over the network using **rmt**(8C). Since **restore** is normally run by root, the name of the local machine must appear in the **.rhosts** file of the remote machine. If the file is specified as *user*@*machine*:*device*, **restore** will attempt to execute as the specified user on the remote machine. The specified user must have a **.rhosts** file on the remote machine that allows root from the local machine. If **restore** is called as **rrestore,** the tape defaults to **dumphost:/dev/rmt8.** To direct the input from a desired remote machine, set up an alias for **dumphost** in the file **/etc/hosts.**

**s** *n* Skip to the *n*'th file when there are multiple dump files on the same tape. For example, the command:

**example# restore xfs /dev/nrar0 5**

would position you at the fifth file on the tape.

## USAGE

### Interactive Commands

**restore** enters interactive mode when invoked with the **i** option. Interactive commands are reminiscent of the shell. For those commands that accept an argument, the default is the current directory.

**ls** [*directory*] List files in *directory* or the current directory, represented by a '.' (period). Directories are appended with a '/' (backslash). Entries marked for extraction are prefixed with a '*' (asterisk). If the verbose option is in effect, inode numbers are also listed.

**cd** *directory*
Change to directory *directory* (within the dump-tape).

**pwd** Print the full pathname of the current working directory.

**add** [*filename*]
Add the current directory, or the named file or directory **directory** to the list of files to extract. If a directory is specified, add that directory and its files (recursively) to the extraction list (unless the **h** modifier is in effect).

**delete** [*filename*]
Delete the current directory, or the named file or directory from the list of files to extract. If a directory is specified, delete that directory and all its descendents from the extraction list (unless the **h** modifier is in effect). The most expedient way to extract a majority of files from a directory is to add that directory to the extraction list, and then delete specific files to omit.

**extract** Extract all files on the extraction list from the dump tape. **restore** asks which volume the user wishes to mount. The fastest way to extract a small number of files is to start with the last tape volume and work toward the first.

**verbose** Toggle the status of the **v** modifier. While **v** is in effect, the **ls** command lists the inode numbers of all entries, and **restore** displays information about each file as it is extracted.

**help** Display a summary of the available commands.

**quit** **restore** exits immediately, even if the extraction list is not empty.

FILES

| | |
|---|---|
| **/dev/rmt8** | the default tape drive |
| **dumphost:/dev/rmt8** | the default tape drive if called as **rrestore** |
| **/tmp/rstdir*** | file containing directories on the tape |
| **/tmp/rstmode*** | owner, mode, and timestamps for directories |
| **./restoresymtable** | information passed between incremental restores |

SEE ALSO

**dump**(8), **mkfs**(8), **mount**(8), **newfs**(8), **rmt**(8C)

BUGS

**restore** can get confused when doing incremental restores from dump tapes that were made on active file systems.

A "level **0**" dump must be done after a full restore. Because **restore** runs in user mode, it has no control over inode allocation; this means that **restore** repositions the files, although it does not change their contents. Thus, a full dump must be done to get a new set of directories reflecting the new file positions, so that later incremental dumps will be correct.

DIAGNOSTICS

**restore** complains about bad option characters.

Read errors result in complaints. If **y** has been specified, or the user responds **y**, **restore** will attempt to continue.

If the dump extends over more than one tape, **restore** asks the user to change tapes. If the **x** or **i** option has been specified, **restore** also asks which volume the user wishes to mount.

There are numerous consistency checks that can be listed by **restore**. Most checks are self-explanatory or can "never happen". Common errors are given below.

**Converting to new file system format.**
A dump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

*filename*: **not found on tape**
The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a dump tape created on an active file system.

**expected next file** *inumber*, **got** *inumber*
A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

**Incremental tape too low**
When doing an incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

**Incremental tape too high**
When doing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or one that has too high an incremental level has been loaded.

**Tape read error while restoring** *filename*
**Tape read error while skipping over inode inumber**
**Tape read error while trying to resynchronize**
**A tape read error has occurred.**

If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

**resync restore, skipped** *num* **blocks**

After a tape read error, **restore** may have to resynchronize itself. This message lists the number of blocks that were skipped over.

## NAME

rexd – RPC-based remote execution server

## SYNOPSIS

**/usr/etc/rpc.rexd**

## DESCRIPTION

**rexd** is the Sun RPC server for remote program execution. This daemon is started by **inetd**(8) whenever a remote execution request is made,

For noninteractive programs, the standard file descriptors are connected directly to TCP connections. Interactive programs involve pseudo-terminals, in a fashion that is similar to the login sessions provided by **rlogin**(1). This daemon may use the NFS to mount file systems specified in the remote execution request.

## FILES

| | |
|---|---|
| **/dev/ttyp***n* | pseudo-terminals used for interactive mode |
| **/etc/passwd** | authorized users |
| **/tmp/dbrexd??????** | temporary mount points for remote file systems. |

## SEE ALSO

**on**(1), **rex**(3), **exports**(5), **inetd**(8), **inetd.conf**(5)

## DIAGNOSTICS

Diagnostic messages are normally printed on the console, and returned to the requestor.

## BUGS

Should be better access control.

## RESTRICTIONS

Root cannot execute commands using **rexd** client programs such as **on**(1).

NAME

rexecd – remote execution server

SYNOPSIS

**/usr/etc/in.rexecd** *host.port*

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**rexecd** is the server for the **rexec**(3N) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords. It is invoked automatically as needed by **inetd**(8C), and then executes the following protocol:

1) The server reads characters from the socket up to a null (\0) byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) **rexecd** then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **rexecd**.

SEE ALSO

**inetd**(8C)

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

**username too long**
The name is longer than 16 characters.

**password too long**
The password is longer than 16 characters.

**command too long**
The command line passed exceeds the size of the argument list (as configured into the system).

**Login incorrect.**
No password file entry for the user name existed.

**Password incorrect.**
The wrong password was supplied.

**No remote directory.**
The **chdir** command to the home directory failed.

**Try again.**
A **fork** by the server failed.

**/usr/bin/sh: ...**

The user's login shell could not be started.

BUGS

Indicating **'Login incorrect'** as opposed to **'Password incorrect'** is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

## NAME

rlogind – remote login server

## SYNOPSIS

**/usr/etc/in.rlogind** *host.port*

## DESCRIPTION

**rlogind** is the server for the **rlogin**(1C) program. The server provides a remote login facility with authentication based on privileged port numbers.

**rlogind** is invoked by **inetd**(8C) when a remote login connection is established, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The client's address and port number are passed as arguments to **rlogind** by **inetd** in the form *host.port* with host in hex and port in decimal.

2) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see **hosts**(5)), the server aborts the connection.

Once the source port and address have been checked, **rlogind** allocates a pseudo-terminal (see **pty**(4)), and manipulates file descriptors so that the slave half of the pseudo-terminal becomes the **stdin**, **stdout**, and **stderr** for a login process. The login process is an instance of the **login**(1) program, invoked with the –r option. The login process then proceeds with the authentication process as described in **rshd**(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseudo-terminal, operating as an intermediary between the login process and the client instance of the **rlogin** program. In normal operation, the packet protocol described in **pty**(4) is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, **TERM**; see **environ**(5V).

## SEE ALSO

**inetd**(8C)

## DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

**Hostname for your address unknown.**
No entry in the host name database existed for the client's machine.

**Try again.**
A *fork* by the server failed.

**/usr/bin/sh:...**
The user's login shell could not be started.

## BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

## NAME

rmail – handle remote mail received via uucp

## SYNOPSIS

**rmail** *recipient*...

## DESCRIPTION

**rmail** interprets incoming mail received through **uucp**(1C), collapsing "From" lines in the form generated by **binmail**(1) into a single line of the form *return-path*!*sender*, and passing the processed mail on to **sendmail**(8).

**rmail** is explicitly designed for use with **uucp**(1C) and **sendmail**(8).

## SEE ALSO

**binmail**(1), **uucp**(1C), **sendmail**(8)

## NAME

rmt – remote magtape protocol module

## SYNOPSIS

**/usr/etc/rmt**

## DESCRIPTION

**rmt** is a program used by the remote dump and restore programs in manipulating a magnetic tape drive through an interprocess communication connection. **rmt** is normally started up with an **rexec**(3N) or **rcmd**(3N) call.

The **rmt** program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

**A***number***\n**

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

**E***error-number***\n***error-message***\n**

where *error-number* is one of the possible error numbers described in **intro**(2) and *error-message* is the corresponding error string as printed from a call to **perror**(3). The protocol is comprised of the following commands (a space is present between each token):

**S** Return the status of the open device, as obtained with a **MTIOCGET ioctl** call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

**C** *device* Close the currently open device. The *device* specified is ignored.

**I***operation count*
Perform a **MTIOCOP ioctl**(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the **ioctl** call. The return value is the *count* parameter when the operation is successful.

**L** *whence offset* Perform an **lseek**(2) operation using the specified parameters. The response value is that returned from the **lseek** call.

**O** *device mode* Open the specified *device* using the indicated *mode*. *device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to **open**(2V). If a device had already been opened, it is closed before a new open is performed.

**R***count* Read *count* bytes of data from the open device. **rmt** performs the requested **read**(2V) and responds with **A***count-read***\n** if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

**W** *count* Write data onto the open device. **rmt** reads *count* bytes from the connection, aborting if a premature EOF is encountered. The response value is that returned from the **write**(2V) call.

Any other command causes **rmt** to exit.

## DIAGNOSTICS

All responses are of the form described above.

## SEE ALSO

**intro**(2), **ioctl**(2), **lseek**(2), **open**(2V), **read**(2V), **write**(2V), **perror**(3), **rcmd**(3N), **rexec**(3N), **mtio**(4), **dump**(8), **restore**(8)

**BUGS**

People tempted to use this for a remote file access protocol are discouraged.

## NAME

route – manually manipulate the routing tables

## SYNOPSIS

**/usr/etc/route** [ **–fhn** ] **add** | **delete** *destination* [ *gateway* [ *metric* ] ]

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**route** manually manipulate the network routing tables normally maintained by the system routing daemon, **routed**(8C). **route** allows the super-user to operate directly on the routing table for the specific host or network indicated by *destination*. The *gateway* argument, if present, indicates the network gateway to which packets should be addressed. The *metric* argument indicates the number of ''hops'' to the *destination*. The default value for *metric* is 0.

The **add** command instructs **route** to add a route to *destination*. **delete** deletes a route.

Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with the *destination* parameter. If the destination has a ''local address part'' of **INADDR_ANY**, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected by a gateway, the *metric* parameter should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up in the hosts database, **/etc/hosts**. If this lookup fails, then the name is looked up in the networks database, **/etc/networks**.

## OPTIONS

**–f** Flush the routing tables of all gateway entries. If this is used in conjunction with one of the subcommands described below, **route** flushes the gateways before preforming the subcommand.

**–h** Treat the destination as a host.

**–n** Treat the destination as a network.

## FILES

**/etc/hosts**
**/etc/networks**

## SEE ALSO

**ioctl**(2), **routing**(4N), **routed**(8C)

## BUGS

The change operation is not implemented, one should add the new route, then delete the old one.

## DIAGNOSTICS

**add** *destination address*:*gateway addresss flags value*
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the **ioctl**(2) call.

**delete** *destination address*:*gateway addresss flags value*
The specified route is being deleted.

*destination address* **done**
When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

**Network is unreachable**
An attempt to add a route failed because the gateway listed was not on a directly-connected network. Give the next-hop gateway instead.

**not in table**
A delete operation was attempted for an entry that is not in the table.

**routing table overflow**
An add operation was attempted, but the system was unable to allocate memory to create the new entry.

**NAME**

routed – network routing daemon

**SYNOPSIS**

**/etc/in.routed** [ **–qstv** ] [ *logfile* ]

**DESCRIPTION**

**routed** is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation **routed** listens on **udp**(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When **routed** is started, it uses the SIOCGIFCONF **ioctl**(2) to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. **routed** then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, **routed** formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

*request* packets received by **routed** are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (that is, the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, **routed** records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. **routed** waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, **routed** also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

Supplying the –s option forces **routed** to supply routing information whether it is acting as an internetwork router or not. The –q option is the opposite of the –s option. If the –t option is specified, all packets sent or received are printed on the standard output. In addition, **routed** will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process. Any other argument supplied is interpreted as the name of file in which **routed**'s actions should be logged. This log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changed route. The -v option allows a logfile to be created showing the changes made to the routing tables with a timestamp.

In addition to the facilities described above, **routed** supports the notion of "distant" *passive* and *active* gateways. When **routed** is started up, it reads the file **/etc/gateways** to find gateways which may not be identified using the SIOGIFCONF **ioctl**. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (that is, they should have a **routed** process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The **/etc/gateways** is comprised of a series of lines, each in the following format:

**< net | host >** *filename1* **gateway** *filename2* **metric** *value* **< passive | active >**

The **net** or **host** keyword indicates if the route is to a network or specific host.

*filename1* is the name of the destination network or host. This may be a symbolic name located in **/etc/networks** or **/etc/hosts**, or an Internet address specified in "dot" notation; see **inet**(3N).

*filename2* is the name or address of the gateway to which messages should be forwarded.

*Value* is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as passive or active (as described above).

## FILES

| | |
|---|---|
| **/etc/gateways** | for distant gateways |
| **/etc/networks** | |
| **/etc/hosts** | |

## SEE ALSO

**ioctl**(2), **inet**(3N), **udp**(4P)

## BUGS

The kernel's routing tables may not correspond to those of **routed** for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

**routed** should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME

rpcinfo – report RPC information

SYNOPSIS

**rpcinfo –p** [ *host* ]

**rpcinfo** [ **–n** *portnum* ] **–u** *host program* [ *version* ]

**rpcinfo** [ **–n** *portnum* ] **–t** *host program* [ *version* ]

**rpcinfo –b** *program version*

**rpcinfo –d** *program version*

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**rpcinfo** makes an RPC call to an RPC server and reports what it finds.

OPTIONS

**–p** Probe the portmapper on *host*, and print a list of all registered RPC programs. If *host* is not specified, it defaults to the value returned by **hostname**(1).

**–u** Make an RPC call to procedure 0 of *program* on the specified *host* using UDP, and report whether a response was received.

**–t** Make an RPC call to procedure 0 of *program* on the specified *host* using TCP, and report whether a response was received.

**–n** Use *portnum* as the port number for the *–t* and *–u* options instead of the port number given by the portmapper.

**–b** Make an RPC broadcast to procedure 0 of the specified *program* and *version* using UDP and report all hosts that respond.

**–d** Delete registration for the RPC service of the specified *program* and *version*. This option can be exercised only by the super-user.

The *program* argument can be either a name or a number.

If a *version* is specified, **rpcinfo** attempts to call that version of the specified *program*. Otherwise, **rpcinfo** attempts to find all the registered version numbers for the specified *program* by calling version 0 (which is presumed not to exist; if it does exist, **rpcinfo** attempts to obtain this information by calling an extremely high version number instead) and attempts to call each registered version. Note: the version number is required for **–b** and **–d** options.

EXAMPLES

To show all of the RPC services registered on the local machine use:

**example% rpcinfo -p**

To show all of the RPC services registered on the machine named **klaxon** use:

**example% rpcinfo -p klaxon**

To show all machines on the local net that are running the Yellow Pages service use:

**example% rpcinfo -b ypserv 'version' | uniq**

where 'version' is the current Yellow Pages version obtained from the results of the **–p** switch above.

To delete the registration for version 1 of the **walld** service use:

**example% rpcinfo -d walld 1**

SEE ALSO

**rpc**(5), **portmap**(8C)

*RPC Programming Guide* in *Network Programming*

BUGS

In releases prior to SunOS 3.0, the Network File System (NFS) did not register itself with the portmapper; **rpcinfo** cannot be used to make RPC calls to the NFS server on hosts running such releases.

## NAME

rquotad – remote quota server

## SYNOPSIS

**/usr/etc/rpc.rquotad**

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rquotad** is an **rpc**(4) server which returns quotas for a user of a local file system which is mounted by a remote machine over the NFS. The results are used by **quota**(1) to display user quotas for remote file systems. The **rquotad** daemon is normally invoked by **inetd**(8C).

## FILES

| | |
|---|---|
| **quotas** | quota file at the file system root |

## SEE ALSO

**quota**(1), **nfs**(4), **rpc**(4), **services**(5), **inetd**(8C)

NAME

rshd – remote shell server

SYNOPSIS

**/usr/etc/in.rshd** *host.port*

DESCRIPTION

**rshd** is the server for the **rcmd**(3N) routine and, consequently, for the **rsh**(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

**rshd** is invoked by **inetd**(8C) each time a shell service is requested, and executes the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The clients host address (in hex) and port number (in decimal) are the argument passed to **rshd.**

2) The server reads characters from the socket up to a null (\0) byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see **hosts**(5)), the server aborts the connection.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) **rshd** then validates the user according to the following steps. The remote user name is looked up in the password file and a **chdir** is performed to the user's home directory. If the lookup or fails, the connection is terminated. If the **chdir** fails, it does a **chdir** to / **(root).** If the user is not the super-user, (user ID 0), the file **/etc/hosts.equiv** is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file **.rhosts** in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **rshd.**

FILES

**/etc/hosts.equiv**

SEE ALSO

**rsh**(1C), **rcmd**(3N), **syslogd**(8)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

DIAGNOSTICS

The following diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

**locuser too long**
The name of the user on the client's machine is longer than 16 characters.

**remuser too long**
The name of the user on the remote machine is longer than 16 characters.

**command too long**
The command line passed exceeds the size of the argument list (as configured into the system).

**Hostname for your address unknown.**
No entry in the host name database existed for the client's machine.

**Login incorrect.**
No password file entry for the user name existed.

**Permission denied.**
The authentication procedure described above failed.

**Can't make pipe.**
The pipe needed for the **stderr**, was not created.

**Try again.**
A *fork* by the server failed.

**/usr/bin/sh:...**
The user's login shell could not be started.

In addition, daemon's status messages and internal diagnostics are logged to the appropriate system log using the **syslogd**(8) facility.

## NAME

rstatd – kernel statistics server

## SYNOPSIS

**/usr/etc/rpc.rstatd**

## DESCRIPTION

**rstatd** is a server which returns performance statistics obtained from the kernel. These statistics are graphically displayed by **perfmeter**(1). The **rstatd** daemon is normally invoked by **inetd**(8C).

Systems with disk drivers to be monitored by this daemon must be configured so as to report disk (**_dk_xfer**) statistics.

## SEE ALSO

**perfmeter**(1), **services**(5), **inetd**(8C)

**NAME**

rusersd – network username server

**SYNOPSIS**

**/usr/etc/rpc.rusersd**

**AVAILABILITY**

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**rusersd** is a server that returns a list of users on the network. The **rusersd** daemon is normally invoked by **inetd**(8C).

**SEE ALSO**

**perfmeter**(1), **rusers**(1C), **services**(5), **inetd**(8C)

**NAME**

rwalld – network rwall server

**SYNOPSIS**

**/usr/etc/rpc.rwalld**

**AVAILABILITY**

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**rwalld** is a server that handles **rwall**(1C) and **shutdown**(2) requests. It is implemented by calling **wall**(1) to all the appropriate network machines. The **rwalld** daemon is normally invoked by **inetd**(8C).

**SEE ALSO**

**rwall**(1C), **wall**(1), **shutdown**(2) **services**(5), **inetd**(8C),

## NAME

rwhod – system status server

## SYNOPSIS

**/usr/etc/rwhod**

## AVAILABILITY

Due to its potential impact on network performance, this service is commented out of the **/etc/rc.local** system initialization script. It is provided only for 4.3 BSD compatibility.

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**rwhod** is the server which maintains the database used by the **rwho**(1C) and **ruptime**(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

**rwhod** operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other **rwhod** servers' status messages, validating them, then recording them in a collection of files located in the directory **/var/spool/rwho**.

The **rwho** server transmits and receives messages at the port indicated in the "rwho" service specification, see **services**(5). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];    /* tty name */
        char    out_name[8];    /* user id */
        long    out_time;       /* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
                struct  whoent {
                struct  outmp we_utmp;
                int     we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the **w**(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the **gethostname**(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the **utmp**(5) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the **rwho** server are discarded unless they originated at a **rwho** server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by **rwhod** are placed in files named **whod.***hostname* in the directory **/var/spool/rwho**. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. **rwhod** performs an **nlist**(3) on **/vmunix** every 10 minutes to guard against the possibility that this file is not the system image currently operating.

**FILES**

**/var/spool/rwho**

**DIAGNOSTICS**

Status and diagnostic messages are logged to the appropriate system log using the **syslogd**(8) facility.

**SEE ALSO**

**rwho**(1C), **ruptime**(1C), **w**(1), **gethostname**(2), **nlist**(3), **utmp**(5), **syslogd**(8)

**BUGS**

This service takes up progressively more network bandwidth as the number of hosts on the local net increases. For large networks, the cost becomes prohibitive. RPC-based services such as **rup**(1C) and **rusers**(1C) provide a similar function with greater efficiency.

**rwhod** should relay status information between networks. People often interpret the server dying as a machine going down.

NAME

sa, accton – system accounting

SYNOPSIS

**/usr/etc/sa** [ **–abcdDfijkKlmnrstu** ] [ –v[*n*] ] [ **–S** *savacctfile* ] [ **–U** *usracctfile* ] [ *filename* ]

**/usr/etc/accton** [ *filename* ]

DESCRIPTION

With an argument naming an existing *filename*, **accton** causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

**sa** reports on, cleans up, and generally maintains accounting files.

**sa** is able to condense the information in **/var/adm/acct** into a summary file **/var/adm/savacct** which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system **/var/adm/acct** can grow by 500K bytes per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; **/var/adm/acct** is the default.

Output fields are labeled: **cpu** for the sum of user+system time (in minutes), **re** for real time (also in minutes), **k** for CPU-time averaged core usage (in 1k units), **avio** for average number of I/O operations per execution. With options fields labeled **tio** for total I/O operations, **k*sec** for CPU storage integral (kilo-core seconds), **u** and **s** for user and system CPU time alone (both in minutes) will sometimes appear.

**sa** also breaks out accounting statistics by user. This information is kept in the file **/var/adm/usracct**.

OPTIONS

**–a** Print all command names, even those containing unprintable characters and those used only once. By default, those are placed under the name '***other.'

**–b** Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

**–c** Besides total user, system, and real time for each command print percentage of total time over all commands.

**–d** Sort by average number of disk I/O operations.

**–D** Print and sort by total number of disk I/O operations.

**–f** Force no interactive threshold compression with **–v** flag.

**–i** Do not read in summary file.

**–j** Instead of total minutes time for each category, give seconds per call.

**–k** Sort by CPU-time average memory usage.

**–K** Print and sort by CPU-storage integral.

**–l** Separate system and user time; normally they are combined.

**–m** Print number of processes and number of CPU minutes for each user.

**–n** Sort by number of calls.

**–r** Reverse order of sort.

**–s** Merge accounting file into summary file **/var/adm/savacct** when done.

**–t** For each command report ratio of real time to the sum of user and system times.

**–u** Superseding all other flags, print for each record in the accounting file the user ID and command name.

**–v** Followed by a number *n*, types the name of each command used *n* times or fewer. If *n* is not specified, it defaults to 1. Await a reply from the terminal; if it begins with **y**, add the command to the category '**junk**.' This is used to strip out garbage.

**–S** The following filename is used as the command summary file instead of **/var/adm/savacct**.

**–U** The following filename is used instead of **/var/adm/usracct** to accumulate the per-user statistics printed by the **–m** option.

**FILES**

| | |
|---|---|
| **/var/adm/acct** | raw accounting |
| **/var/adm/savacct** | summary by command |
| **/var/adm/usracct** | summary by user ID |

**SEE ALSO**

**acct**(2), **acct**(5), **ac**(8)

**BUGS**

**sa**'s execution time increases linearly with the magnitude of the largest positive user ID in **/etc/passwd**.

NAME

savecore – save a core dump of the operating system

SYNOPSIS

**/usr/etc/savecore** *dirname* [ *system-name* ]

DESCRIPTION

**savecore** saves a core dump of the kernel (assuming that one was made) and writes a reboot message in the shutdown log. It is meant to be called near the end of the **/etc/rc.local** file after the system boots. However, it is not normally run by default. You must edit that file to enable it.

**savecore** checks the core dump to be certain it corresponds with the version of the operating system currently running. If it does, **savecore** saves the core image in the file *dirname*/**vmcore.***n* and the kernel's namelist, in *dirname*/**vmunix.***n* The trailing *.n* in the pathnames is replaced by a number which grows every time **savecore** is run in that directory.

Before **savecore** writes out a core image, it reads a number from the file *dirname*/**minfree**. If there is less free space on the filesystem containing *dirname* than the number obtained from the minfree file, the core dump is not saved. If the **minfree** file does not exist, **savecore** always writes out the core file (assuming that a core dump was taken).

**savecore** also logs a reboot message using facility LOG_AUTH (see **syslog**(3)) If the system crashed as a result of a panic, **savecore** logs the panic string too.

If the core dump was from a system other than **/vmunix**, the name of that system must be supplied as *system-name*.

FILES

| | |
|---|---|
| **/vmunix** | the kernel |
| **/etc/rc.local** | |

SEE ALSO

**syslog**(3), **sa**(8), **crash**(8S)

BUGS

Can be fooled into thinking a core dump is the wrong size.

You must run **savecore** very soon after booting — before the swap space containing the crash dump is overwritten by programs currently running.

NAME

sendmail – send mail over the internet

SYNOPSIS

**/etc/sendmail** [ **–ba** ] [ **–bd** ] [ **–bi** ] [ **–bm** ] [ **–bp** ] [ **–bs** ] [ **–bt** ] [ **–bv** ] [ **–bz** ]
[ **–C***file* ] [ **–d***X* ] [ **–F***fullname* ] [ **–f***name* ] [ **–h***N* ] [ **–n** ] [ **–o***x value* ] [ **–q**[ *time* ] ]
[ **–r***name* ] [ **–t** ] [ **–v** ] [ *address* ... ]

DESCRIPTION

**sendmail** sends a message to one or more people, routing the message over whatever networks are necessary. **sendmail** does internetwork forwarding as necessary to deliver the message to the correct place.

**sendmail** is not intended as a user interface routine; other programs provide user-friendly front ends; **sendmail** is used only to deliver pre-formatted messages.

With no flags, **sendmail** reads its standard input up to an EOF, or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Local addresses are looked up in the local **aliases**(5) file, or by using the Yellow Pages name service, and aliased appropriately. In addition, if there is a **.forward** file in a recipient's home directory, **sendmail** forwards a copy of each message to the list of recipients that file contains. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in alias expansions, for example, if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

**sendmail** will also route mail directly to other known hosts in a local network. The list of hosts to which mail is directly sent is maintained in the file **/usr/lib/mailhosts.**

OPTIONS

**–ba** Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.

**–bd** Run as a daemon, waiting for incoming SMTP connections.

**–bi** Initialize the alias database.

**–bm** Deliver mail in the usual way (default).

**–bp** Print a summary of the mail queue.

**–bs** Use the SMTP protocol as described in RFC 821. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.

**–bt** Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

**–bv** Verify names only — do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

**–bz** Create the configuration freeze file.

**–C***file* Use alternate configuration file.

**–d***X* Set debugging value to *X*.

**–F***fullname*
Set the full name of the sender.

**–f***name* Sets the name of the "from" person (that is, the sender of the mail). **–f** can only be used by "trusted" users (who are listed in the config file).

**–h***N* Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop.

**–M***id* Attempt to deliver the queued message with message-id **id**.

**–n** Don't do aliasing.

**–o***x value*
Set option *x* to the specified *value*. Options are described below.

**–q**[*time*]
Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *Time* is given as a tagged number, with **s** being seconds, **m** being minutes, **h** being hours, **d** being days, and **w** being weeks. For example, **–q1h30m** or **–q90m** would both set the timeout to one hour thirty minutes.

**–r***name* An alternate and obsolete form of the **–f** flag.

**–R***string*
Go through the queue of pending mail and attempt to deliver any message with a recipient containing the specified string. This is useful for clearing out mail directed to a machine which has been down for awhile.

**–t** Read message for recipients. "To:", "Cc:", and "Bcc:" lines will be scanned for people to send to. The "Bcc:" line will be deleted before transmission. Any addresses in the argument list will be suppressed.

**–v** Go into verbose mode. Alias expansions will be announced, etc.

PROCESSING OPTIONS

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **–o** flag or in the configuration file. These are described in detail in the *Installation and Operation Guide*. The options are:

A*file* Use alternate alias file.

c On mailers that are considered "expensive" to connect to, do not initiate immediate connection. This requires queueing.

d*x* Set the delivery mode to *x*. Delivery modes are **i** for interactive (synchronous) delivery, **b** for background (asynchronous) delivery, and **q** for queue only — that is, actual delivery is done the next time the queue is run.

D Run **newaliases**(8) to automatically rebuild the alias database, if necessary.

e*x* Set error processing to mode *x*. Valid modes are **m** to mail back the error message, **w** to "write" back the error message (or mail it back if the sender is not logged in), **p** to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes **m** or **w** and if the sender is local to this machine, a copy of the message is appended to the file **dead.letter** in the sender's home directory.

F*mode* The mode to use when creating temporary files.

f Save UNIX-system-style "From" lines at the front of messages.

g*N* The default group ID to use when calling mailers.

H*file* The SMTP help file.

i Do not take dots on a line by themselves as a message terminator.

L*n* The log level.

m Send to "me" (the sender) also if I am in an alias expansion.

o If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (that is, commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.

*Qqueuedir*
Select the directory in which to queue messages.

*rtimeout*
The timeout on reads; if none is set, **sendmail** will wait forever for a mailer.

*Sfile* Save statistics in the named file.

s Always instantiate the queue file, even under circumstances where it is not strictly necessary.

T*time* Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, they will be returned to the sender. The default is three days.

t*stz,dtz* Set the name of the time zone.

u*N* Set the default user id for mailers.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep **sendmail** from suppressing the blanks from between arguments.

**sendmail** returns an exit status describing what it did. The codes are defined in *<sysexits.h>*

| | |
|---|---|
| EX_OK | Successful completion on all addresses. |
| EX_NOUSER | User name not recognized. |
| EX_UNAVAILABLE | Catchall meaning necessary resources were not available. |
| EX_SYNTAX | Syntax error in address. |
| EX_SOFTWARE | Internal software error, including bad arguments. |
| EX_OSERR | Temporary operating system error, such as "cannot fork". |
| EX_NOHOST | Host name not recognized. |
| EX_TEMPFAIL | Message could not be sent immediately, but was queued. |

If invoked as *newaliases*, **sendmail** rebuilds the alias database. If invoked as *mailq*, **sendmail** prints the contents of the mail queue.

## FILES

Except for **/etc/sendmail.cf**, these pathnames are all specified in **/etc/sendmail.cf**. Thus, these values are only approximations.

| | |
|---|---|
| **/etc/aliases** | raw data for alias names |
| **/etc/aliases.pag** | data base of alias names |
| **/etc/aliases.dir** | |
| **/usr/lib/mailhosts** | list of hosts to which mail can be sent directly |
| **/etc/sendmail.cf** | configuration file |
| **/etc/sendmail.fc** | frozen configuration |
| **/etc/sendmail.hf** | help file |
| **/etc/sendmail.st** | collected statistics |
| **/usr/bin/uux** | to deliver uucp mail |
| **/usr/bin/mail** | to deliver local mail |
| **/var/spool/mqueue/*** | temp files and queued mail |
| **~/.forward** | list of recipients for forwarding messages |

## SEE ALSO

**biff(1), binmail(1), mail(1), aliases(5)**

*System and Network Administration*

Su, Zaw-Sing, and Jon Postel, *The Domain Naming Convention for Internet User Applications*, RFC 819, Network Information Center, SRI International, Menlo Park, Calif., August 1982.

Postel, Jon, *Simple Mail Transfer Protocol*, RFC 821, Network Information Center, SRI International, Menlo Park, Calif., August 1982.

Crocker, Dave, *Standard for the Format of ARPA-Internet Text Messages*, RFC 822, Network Information Center, SRI International, Menlo Park, Calif., August 1982.

NAME

setup_client – create or remove an NFS client

SYNOPSIS

**/usr/etc/install/script/setup_client** *op clientname yp_type swapsize rootpath swappath dumppath homepath execpath arch*

DESCRIPTION

**setup_client** adds an NFS client to a server, or to removes one. It can only be run by the super-user. It is also used by **suninstall**(8).

The *op* argument indicates which operation to perform; it can be either **add** or **remove**, to indicate whether to add or remove a client. *clientname* is the hostname of the client. *yp_type* indicates the type of Yellow Pages server or service to provide to the client, if any; it can be one of **master, slave, client** or **none**. *swapsize* is the number of bytes reserved for client's swap file. *rootpath* is the pathname of parent directory in which various client root directories reside; *rootpath/clientname* is the pathname of the client's root directory. *swappath* is the pathname of parent directory in which various client swap files reside; *swappath/clientname* is the pathname of the client's swap file. *dumppath* is the parent pathname in which various client dump files reside; *dumppath/clientname* is the pathname of the client's dump file. *homepath* is the pathname of the (parent) directory in which the various home directories are to reside; it is the pathname of the directory that the client is to mount as **/home**. *execpath* is the full pathname of the directory in which the executables for the architecture specified by the *arch* argument. This is the directory that the client mounts as **/usr**. *arch* specifies the client's architecture (for instance, **sun4, sun3**...). **setup_client** with no arguments displays a usage message that includes the proper *arch* argument for each supported architecture.

USAGE

Before you add or remove a client, you must first make sure that the Internet and Ethernet addresses for *clientname* are listed in the YP hosts database (if the server is running the YP), or in the server's **/etc/hosts** and **/etc/ethers** databases, respectively (otherwise). Then, run **setup_client** with the **add** or **remove** operation. When adding a client, you must then bootstrap that client machine.

You cannot add a client to a server that does not support the specified architecture. The executable directory for that client's architecture must be present on the server. If this file is absent, an error results.

**setup_client** updates the **/etc/bootparams** file. If the server is a YP master, it updates local YP database. It *does not* propagate the local update to other YP servers. To propagate the updates, use the following commands:

**example# cd /var/yp**
**example# make**

If the server is running YP but is not a YP master, **setup_client** issues a warning to indicate that the database is out of date.

When *arch* is given as **sun2**, **suninstall** issues a reminder to run the **/usr/etc/ndbootd** daemon for booting Sun-2 systems.

**setup_client** creates *swappath/clientname* with the *size*, (number of bytes) you specify. You can append one of **K** or **k** to indicate kilobytes, **M** or **m** to indicate megabytes, or **B** or **b** to indicate 512-byte blocks, to *size*. Otherwise, *size* is taken to indicate an exact byte count.

**suninstall** updates the **/etc/exports** file to allow root access to each client's root file system. It exports the client's swap and dump partitions only to the client. Note: the system administrator should verify that the **/etc/exports** file contains correct information, and that file systems are exported to the correct users and groups. Refer to **exportfs**(8) for details on exporting file systems.

## EXAMPLES

This example shows how to add a Sun-4 system NFS client to a server.

```
example# setup_client add frodo client 16M /exports/roots /exports/swaps /exports/dumps /home \
/exports/execs/sun4/4.0 sun4
```

To remove this client, you would merely substitute **remove** for **add** in the above example.

## FILES

**/etc/hosts**
**/etc/ethers**
**/usr/etc/ndbootd**
**/etc/bootparams**
**/etc/exports**

## SEE ALSO

**exportfs**(8), **setup_exec**(8) **suninstall**(8)

*Installing the SunOS*

## DIAGNOSTICS

**incorrect number of arguments**
Check number and order of the arguments.

**must be run as root (super-user).**
You must be root to use **setup_client**.

**invalid operation type "***xx***".**
Valid operations are **add** and **remove**.

**ATTENTION: *xxxxxxxx* -> boot.sun*?* not created.**
(Sun-3 systems only.) A symbolic link can not be created because the boot file does not exist.

**ATTENTION: *xxxxxxxx*.SUN*?* -> boot.sun*?* not created.**
(Other than Sun-3 systems.) A symbolic link can not be created because the boot file does not exist.

**ATTENTION: /usr/etc/ndbootd needs to be running on server before bringing up "*client*".**
The Sun-2 system boot daemon must be running in order to bootstrap a Sun-2 system.

## NAME

setup_exec – install architecture-dependent executables on a heterogeneous file server

## SYNOPSIS

**/usr/etc/install/setup_exec** *arch execpath*

## DESCRIPTION

**setup_exec** installs architecture-dependent executables from either from a local tape drive or a remote host. It is used to convert a standalone system or homogeneous file server to a heterogeneous file server. **setup_exec** is a forms-based utility that can be invoked directly, but it is also used by **suninstall**(8). It can only be invoked by the super-user.

The *arch* argument specifies the machine archictecture to install (for instance, **sun4, sun3**...). When run with no arguments, **setup_exec** displays a usage line that includes the proper format of the *arch* argument for each supported architecture. *execpath* is the full pathname of the directory in which to install the executables. When **setup_exec** is done, the *execpath* directory is ready to mount as **/usr** by the heterogeneous server's NFS clients of the indicated *arch*.

**setup_exec** also updates the **/etc/exports** file (see **exportfs**(8)) to export the executable directories it has installed. The system administrator should verify this file to make sure that the directory has been exported to the correct groups.

## EXAMPLE

This example shows how to install a directory of executables for Sun-4 system clients running 4.0.

```
example# setup_exec sun4 /exports/execs/sun4/4.0
```

## FILES

| | |
|---|---|
| **/etc/hosts** | hosts database |
| **/etc/ethers** | database of hostnames and Ethernet addresses |
| **/etc/exports** | database of exported file systems |
| **/usr/etc/install/files/extractlist.***arch* | record of extracted categories for the indicated architecture |

## SEE ALSO

**exportfs**(8), **setup_client**(8), **suninstall**(8)

*Installing the SunOS*

## DIAGNOSTICS

**incorrect number of arguments**
Check the number and the order of arguments.

**invalid architecture type "***arch***".**
You supplied a value for *arch* that is not supported.

**invalid tape drive type "***drive***".**
Valid tape drive types are **local** and **remote**.

**invalid tape type "***tape***".**
Valid tape types are **ar**, **st**, **mt**, and **xt**.

**can't reach tapehost "***tapehost***".**
The IP address of *tapehost* is not in the hosts database, that is, the hosts YP database if the Yellow Pages are running, or the **/etc/hosts** file otherwise.

**Load release tape *n***
Mount the release tape specified on the screen and type RETURN to continue.

## NAME

showmount – show all remote mounts

## SYNOPSIS

**/usr/etc/showmount** [ **–ade** ] [ host ]

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**showmount** lists all the clients that have remotely mounted a filesystem from *host*. This information is maintained by the **mountd**(8C) server on *host*, and is saved across crashes in the file **/etc/rmtab**. The default value for *host* is the value returned by **hostname**(1).

## OPTIONS

**–a** Print all remote mounts in the format

*hostname*:*directory*

where **hostname** is the name of the client, and **directory** is the root of the file system that has been mounted.

**–d** List directories that have been remotely mounted by clients.

**–e** Print the list of exported file systems.

## FILES

**/etc/rmtab**

## SEE ALSO

**hostname**(1), **exports**(5), **exports**(5), **mountd**(8C)

## BUGS

If a client crashes, its entry will not be removed from the list until it reboots and executes '**umount –a**'.

## NAME

shutdown – close down the system at a given time

## SYNOPSIS

**/usr/etc/shutdown** [ **–fhknr** ] [ *time* [ *warning-message* ... ]

## DESCRIPTION

**shutdown** provides an automated procedure to notify users when the system is to be shut down. *time* specifies when **shutdown** will bring the system down; it may be the word **now** (indicating an immediate shutdown), or it may specify a future time in one of two formats: *+number* and *hour:min*. The first form brings the system down in *number* minutes, and the second brings the system down at the time of day indicated in 24-hour notation.

At intervals that get closer as the apocalypse approaches, warning messages are displayed at terminals of all logged-in users, and of users who have remote mounts on that machine. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating **/etc/nologin** and writing a message there. If this file exists when a user attempts to log in, **login**(1) prints its contents and exits. The file is removed just before **shutdown** exits.

At shutdown time a message is written to the system log daemon, **syslogd**(8), containing the time of shutdown, the instigator of the shutdown, and the reason. Then a terminate signal is sent to **init**, which brings the system down to single-user mode.

The time of the shutdown and the warning message are placed in **/etc/nologin**, which should be used to inform the users as to when the system will be back up, and why it is going down (or anything else).

## OPTIONS

As an alternative to the above procedure, these options can be specified:

**–f** Arrange, in the manner of **fastboot**(8), that when the system is rebooted, the file systems will not be checked.

**–h** Execute **halt**(8).

**–k** Simulate shutdown of the system. Do not actually shut down the system.

**–n** Prevent the normal **sync**(2) before stopping.

**–r** Execute **reboot**(8).

## FILES

| | |
|---|---|
| **/etc/nologin** | tells login not to let anyone log in |
| **/etc/xtab** | list of remote hosts that have mounted this host |

## SEE ALSO

**login**(1), **sync**(2), **fastboot**(8), **halt**(8), **reboot**(8), **syslogd**(8)

## BUGS

Only allows you to bring the system down between "now" and 23:59 if you use the absolute time for shutdown.

## NAME

spray – spray packets

## SYNOPSIS

**/usr/etc/spray** *host* [ **–c** *count* ] [ **–d** *delay* ] [ **–i** *delay* ] [ **–l** *length* ] *host*

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**spray** sends a one-way stream of packets to *host* using RPC, and reports how many were received, as well as the the transfer rate. The *host* argument can be either a name or an internet address.

## OPTIONS

**–c** *count*

Specify how many packets to send. The default value of *count* is the numbers of packets required to make the total stream size 100000 bytes.

**–d** *delay*

Specify how may microseconds to pause between sending each packet. The default is 0.

**–i** *delay* Use ICMP echo packets rather than RPC. Since ICMP automatically echos, this creates a two way stream.

**–l** *length*

The *length* parameter is the numbers of bytes in the ethernet packet that holds the RPC call message. Since the data is encoded using XDR, and XDR only deals with 32 bit quantities, not all values of *length* are possible, and **spray** rounds up to the nearest possible value. When *length* is greater than 1514, then the RPC call can no longer be encapsulated in one Ethernet packet, so the *length* field no longer has a simple correspondence to Ethernet packet size. The default value of *length* is 86 bytes (the size of the RPC and UDP headers)

## SEE ALSO

**icmp**(4P), **ping**(8C), **sprayd**(8C)

**NAME**

sprayd – spray server

**SYNOPSIS**

**/usr/etc/rpc.sprayd**

**AVAILABILITY**

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

**DESCRIPTION**

**rpc.sprayd** is a server which records the packets sent by **spray**(8C). The **rpc.sprayd** daemon is normally invoked by **inetd**(8C).

**SEE ALSO**

**inetd**(8C), **spray**(8C)

NAME

statd – network status monitor

SYNOPSIS

**/etc/rpc.statd**

DESCRIPTION

**statd** is an intermediate version of the status monitor. It interacts with **lockd**(8C) to provide the crash and recovery functions for the locking services on NFS.

FILES

**/etc/sm**
**/etc/sm.bak**
**/etc/state**

SEE ALSO

**statmon**(5), **lockd**(8C)

BUGS

The crash of a site is only detected upon its recovery.

## NAME

sticky – persistent text and append-only directories

## DESCRIPTION

The *sticky bit* (file mode bit 01000, see **chmod**(2)) is used to indicate special treatment for certain executable files and directories.

### Sticky Text Executable Files

While the sticky bit is set on a sharable executable file, the text of that file will not be removed from the system swap area. Thus the file does not have to be fetched from the file system upon each execution. As long as a copy remains in the swap area, the original text cannot be overwritten in the file system, nor can the file be deleted. Directory entries can be removed so long as one link remains.

Sharable executable files are made by the **–n** and **–z** options of **ld**(1).

To replace a sticky file that has been used:

1. Clear the sticky bit with **chmod**(1V).
2. Execute the old program to flush the swapped copy. This can be done safely even if others are using it.
3. Overwrite the sticky file. If the file is being executed by any process, writing will be prevented; it suffices to simply remove the file and then rewrite it, being careful to reset the owner and mode with **chmod** and **chown**(2).
4. Set the sticky bit once again, if still needed.

Only the super-user can set the sticky bit on a sharable executable file.

### Sticky Directories

A directory for which the sticky bit is set restricts deletion of files it contains. A file in a sticky directory may only be removed or renamed by a user who has write permission on the directory, and either owns the file, owns the directory, or is the super-user. This is useful for directories such as **/tmp**, which must be publicly writable, but should deny users permission to arbitrarily delete or rename the files of others.

Any user may create a sticky directory. See **chmod** for details about modifying file modes.

## BUGS

Since the text areas of sticky text executables are stashed in the swap area, abuse of the feature can cause a system to run out of swap.

Neither **open**(2V) nor **mkdir**(2) will create a file with the sticky bit set.

## FILES

**/tmp**

## SEE ALSO

**chmod**(1V), **ld**(1), **chmod**(2), **chown**(2), **mkdir**(2), **open**(2V)

## NAME

suninstall – install and upgrade the Sun Operating System

## SYNOPSIS

**/usr/etc/install/suninstall**

## DESCRIPTION

**suninstall** is a forms-based subsystem for installing and upgrading the Sun Operating System on Sun-2, Sun-3 and Sun-4 systems. Unlike previous installation subsystems, **suninstall** does not require you to recapitulate an interrupted procedure; it allows you to pick up from where you left off. A new invocation of **suninstall** displays the saved information, and gives you an opportunity to make any needed alterations, before it proceeds.

To abort the installation procedure, use the interrupt character (typically CTRL-C).

**suninstall** allows you to install the operating system onto any system configuration, be it standalone, dataless, a homogeneous file server, or a heterogeneous server. It allows you to install from any distribution tape format, to install the various versions of the operating system needed by clients on a heterogeneous file server; you can install as many different system versions as your disk space may allow.

You can use **suninstall** to convert a 4.0 standalone system into a 4.0 server, without taking down or rebuilding the system. After the initial installation, you can use **setup_client**(8), to add or remove a diskless client while the server is running in multiuser mode. You can use **setup_exec**(8), to convert a 4.0 standalone system or server into a heterogeneous file server while it is running multiuser.

## USAGE

Refer to *Installing the SunOS* for more information on the various menus and selections.

## FILES

| | |
|---|---|
| **/usr/etc/install** | directory containing installation programs, scripts and files |
| **/usr/etc/install/files** | directory containing default data files for clients and hosts |
| **/usr/etc/install/get_*_info** | terminal data-entry forms |
| **/usr/etc/install/installation** | subsystem utility program |
| **/usr/etc/install/makedir** | subsystem utility program |
| **/usr/etc/install/script** | subsystem utility scripts |
| **/usr/etc/install/xdrtoc** | subsystem utility program |

## SEE ALSO

**extract_unbundled**(8), **setup_client**(8), **setup_exec**(8)

*Installing the SunOS*

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

**/usr/etc/swapon –a**

**/usr/etc/swapon** *name...*

DESCRIPTION

**swapon** specifies additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to **swapon** normally occur in the system multi-user initialization file **/etc/rc** making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

The second form gives individual block devices as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

OPTIONS

**–a** Make available all devices of type **swap** in **/etc/fstab**. Using **swapon** with the **–a** option is the normal usage.

FILES

| | |
|---|---|
| **/dev/[ru][pk]?b** | normal paging devices |
| **/etc/fstab** | |
| **/etc/rc** | |

SEE ALSO

**swapon**(2), **init**(8)

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

## NAME

sysdiag – system diagnostics

## SYNOPSIS

**/usr/diag/sysdiag/sysdiag**

## AVAILABILITY

This program is available with the *User Diagnostics* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**sysdiag** is a general-purpose system diagnostic facility that tests the system and reports its findings. It concentrates on three areas of system functionality; memory, peripherals and disk.

To use **sysdiag**, log on as **sysdiag**, then enter the command **sysdiag**.

**sysdiag** creates a **sunview**(1) environment with one window each for memory, peripherals, and disk error messages, plus a window for the console. It also creates date/time and performance monitor graphs. It places abbreviated error messages from the memory, disk, and peripherals in the appropriate windows, and sends console messages to the console window.

When called from a terminal **sysdiag** interleaves all its messages on the screen.

With or without the windows, it places long error messages in files named **log.***xx.nn* where:

*xx* is the name of diagnostic

*nn* is the pass number (increments each pass)

After it completes its test, **sysdiag** displays the error log files by executing the command '**more log***'. These files remain after **sysdiag** exits.

**sysdiag** consists of a user account with a home directory, a collection of scripts, and executable files containing the actual test code.

To configure or change **sysdiag**, either change the shell commands in **/usr/diag/sysdiag/sysdiag**, or change the **sysdiag** user configuration files **.login**, **.sunview**, and **.cshrc**.

## FILES

**.login**
**.sunview**
**.cshrc**

## SEE ALSO

**sunview**(1) See the appropriate diagnostic manual for your Sun system.

## NAME

syslogd – log system messages

## SYNOPSIS

**/usr/etc/syslogd** [ **–d** ] [ **–f***configfile* ] [ **–m** *interval* ]

## DESCRIPTION

**syslogd** reads and forwards system messages to the appropriate log files and/or users, depending upon the priority of a message and the system facility from which it originates. The configuration file **/etc/syslog.conf** (see **syslog.conf**(5)) controls where messages are forwarded. **syslogd** logs a mark (timestamp) message every *interval* minutes (default 20) at priority LOG_INFO to the facility whose name is given as **mark** in the **syslog.conf** file.

A system message consists of a single line of text, which may be prefixed with a priority code number enclosed in angle-brackets (< >); priorities are defined in **<sys/syslog.h>**.

**syslogd** reads from the AF_UNIX address family socket **/dev/log**, from an Internet address family socket specified in **/etc/services**, and from the special device **/dev/klog** (for kernel messages).

**syslogd** reads the configuration file when it starts up, and again whenever it receives a HUP signal, at which time it also closes all files it has open, re-reads its configuration file, and then opens only the log files that are listed in that file. **syslogd** exits when it receives a TERM signal.

As it starts up, **syslogd** creates the file **/etc/syslog.pid**, if possible, containing its process ID.

## Sun386i DESCRIPTION

**syslogd** translates messages using the databases specified on an optional line in the **syslog.conf** as indicated with a **translate** entry.

The format of these databases is described in **translate**(5).

## OPTIONS

| | |
|---|---|
| **–d** | Turn on debugging. |
| **–f***configfile* | Specify an alternate configuration file. |
| **–m** *interval* | Specify an interval, in minutes, between mark messages. |

## FILES

| | |
|---|---|
| **/etc/syslog.conf** | configuration file |
| **/etc/syslog.pid** | process ID |
| **/dev/log** | AF_UNIX address family datagram log socket |
| **/dev/klog** | kernel log device |
| **/etc/services** | network services database |

## SEE ALSO

**logger**(1), **syslog**(3), **syslog.conf**(5)

NAME

talkd – server for talk program

SYNOPSIS

**/usr/etc/in.talkd**

DESCRIPTION

**talkd** is a server used by the **talk**(1) program. It listens at the udp port indicated in the "talk" service description; see **services**(5). The actual conversation takes place on a tcp connection that is established by negotiation between the two machines involved.

SEE ALSO

**talk**(1), **services**(5), **inetd**(8C)

BUGS

The protocol is architecture dependent, and can not be relied upon to work between Sun systems and other machines.

## NAME

telnetd – DARPA TELNET protocol server

## SYNOPSIS

**/usr/etc/in.telnetd**

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**telnetd** is a server which supports the DARPA standard TELNET virtual terminal protocol. **telnetd** is invoked by the internet server (see **inetd**(8C)), normally for requests to connect to the TELNET port as indicated by the **/etc/services** file (see **services**(5)).

**telnetd** operates by allocating a pseudo-terminal device (see **pty**(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as its standard input, output, and error. **telnetd** manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a **TELNET** session is started up, **telnetd** sends TELNET options to the client side indicating a willingness to do *remote echo* of characters, to *suppress go ahead*, and to receive *terminal type information* from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS, ICRNL, and ONLCR enabled (see **termio**(4)).

**telnetd** is willing to *do*: *echo*, *binary*, *suppress go ahead*, and *timing mark*. **telnetd** is willing to have the remote client *do*: *binary*, *terminal type*, and *suppress go ahead*.

## SEE ALSO

**telnet**(1C)

Postel, Jon, and Joyce Reynolds, "Telnet Protocol Specification," RFC 854, Network Information Center, SRI International, Menlo Park, Calif., May 1983.

## BUGS

Some TELNET commands are only partially implemented.

The TELNET protocol allows for the exchange of the number of lines and columns on the user's terminal, but **telnetd** doesn't make use of them.

Because of bugs in the original 4.2 BSD **telnet**(1C), **telnetd** performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2 BSD **telnet**(1C).

Binary mode has no common interpretation except between similar operating systems

The terminal type name received from the remote client is converted to lower case.

The *packet* interface to the pseudo-terminal (see **pty**(4)) should be used for more intelligent flushing of input and output queues.

**telnetd** never sends TELNET *go ahead* commands.

**telnetd** can only support 64 pseudo-terminals.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

**/usr/etc/in.tftpd** [–s] [ *homedir* ]

Sun386i SYNOPSIS

**/usr/etc/in.tftpd** [–s] [–p] [ *homedir* ]

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**tftpd** is a server that supports the DARPA Trivial File Transfer Protocol (TFTP). This server is normally started by **inetd**(8C) and operates at the port indicated in the **tftp** Internet service description in the **/etc/inetd.conf** file; see **inetd.conf**(5) for details.

Before responding to a request, the server attempts to change its current directory to *homedir*; the default value is **/tftpboot**.

Sun386i DESCRIPTION

The **tftpd** daemon acts as described above, except that it will perform certain filename mapping operations unless instructed otherwise by the –p command line argument or when operating in a secure environment. This mapping affects only TFTP boot requests and will not affect requests for existing files.

The semantics of the changes are as follows. Only filenames of the format *ip-address* or *ip-address.arch*, where *ip-address* is the IP address in hex, and *arch* is the hosts's architecture (as returned by the **arch**(1) command), that do not correspond to files in **/tftpboot**, are mapped. If the address is known through a YP lookup, any file of the form **/tftpboot/ip-address*** (with or without a suffix) is returned. If there are multiple such files, any one may be returned. If the *ip-address* is unknown (that is if the **ipalloc** (8C) service says the name service does not know the address), the filename is mapped as follows: Names without the *arch* suffix are mapped into the name **pnp.SUN3**, and names with the suffix are mapped into **pnp.**Arch. That file is returned if it exists.

OPTIONS

–s Secure. When specified, the directory change must succeed; and the daemon also changes its root directory to *homedir*.

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Files may be written only if they already exist and are publicly writable. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling this service.

**tftpd** runs with the user ID and group ID set to –2, under the assumption that no files exist with that owner or group. However, nothing checks this assumption or enforces this restriction.

Sun386i OPTIONS

–p Disable pnp entirely. Do not map filenames.

Sun386i FILES

/tftpboot/* filenames are IP addresses

SEE ALSO

**ipallocd**(8C), **netconfig**(8C), **inetd**(8C), **tftp**(1C)

Sollins, K.R., *The TFTP Protocol (Revision 2)*, RFC 783, Network Information Center, SRI International, Menlo Park, Calif., June 1981.

**Sun386i WARNINGS**

A request for an *ip-address* from a Sun-4 can be satisfied by a file named *ip-address*.**386** for compatibility with some early Sun-4 PROM monitors.

## NAME

tic – terminfo compiler

## SYNOPSIS

**tic** [ –v[*n*] ] [–**c**] *filename*

## DESCRIPTION

Note: Optional Software (System V Option). Refer to *Installing the SunOS* for information on how to install this command.

**tic** compiles a **terminfo**(5V) source file into the compiled format. The results are placed in the directory **/usr/share/lib/terminfo**. The compiled format is used by the **curses**(3V) library.

Each entry in the file describes the capabilities of a particular terminal. When a **use**=*entry* field is given in a terminal entry, **tic** reads in the binary (compiled) description of the indicated *entry* from **/usr/share/lib/terminfo** to duplicate the contents of that entry within the one being compiled. However, if an *entry* by that name is specified in *filename*, the entry in that source file is used first. Also, if a capability is defined in both entries, the definition in the current entry's source file is used.

If the environment variable **TERMINFO** is set, that directory is searched and written to instead of **/usr/share/lib/terminfo**.

## OPTIONS

–v[*n*]
: Verbose. Display trace information on the standard error. The optional integer argument is a number from 1 to 10, inclusive, indicating the desired level of detail. If *n* is omitted, the default is 1.

–**c**
: Only check *filename* for errors. Errors in **use**= links are not detected.

## FILES

**/usr/share/lib/terminfo/?/***
: compiled terminal description data base

## SEE ALSO

**fork**(2), **curses**(3V), **curses**(3X), **malloc**(3), **term**(5), **terminfo**(5V)

## BUGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 1024 bytes.

When the –**c** option is used, duplicate terminal names will not be diagnosed; however, when –**c** is not used, they will be.

For backward compatibility, cancelled capabilities will not be marked as such within the terminfo binary unless the entry name has a '+' within it. Such terminal names are only used for inclusion with a **use**= field, and typically aren't used for actual terminal names.

## DIAGNOSTICS

Most diagnostic messages produced by **tic** are preceded with the approximate line number and the name of the entry being processed.

**mkdir** *name* **returned bad status**
: The named directory could not be created.

**File does not start with terminal names in column one**
: The first thing seen in the file, after comments, must be the list of terminal names.

**Token after a seek(2) not NAMES**
: Somehow the file being compiled changed during the compilation.

**Not enough memory for use_list element**
**Out of memory**
: Not enough free memory was available (*malloc*(3) failed).

**Can't open** *filename*
The named file could not be opened or created.

**Error in writing** *filename*
The named file could not be written to.

**Can't***link* **filename** *to* **filename**
A link failed.

**Error in re-reading compiled** *filename*
The compiled file could not be read back in.

**Premature EOF**
The current entry ended prematurely.

**Backspaced off beginning of line**
This error indicates something wrong happened within **tic**.

**Unknown Capability** – *filename*
The named invalid capability was found within the file.

**Wrong type used for capability ...**
For example, a string capability was given a numeric value.

**Unknown token type**
Tokens must be followed by '@' to cancel, ',' for booleans, '#' for numbers, or '=' for strings.

*name*: **bad term name**
**Line** *n*: **Illegal terminal name** – *name*
**Terminal names must start with a letter or digit**
The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

*name*: **terminal name too long.**
An extremely long terminal name was found.

*name*: **terminal name too short.**
A one-letter name was found.

*name* **defined in more than one entry. Entry being used is** *name* .
An entry was found more than once.

**Terminal name** *name* **synonym for itself**
A name was listed twice in the list of synonyms.

**At least one synonym should begin**
At least one of the names of the terminal should begin with a letter.

*Illegal character* – **c**
The given invalid character was found in the input file.

**Newline in middle of terminal name**
The trailing comma was probably left off of the list of names.

**Missing comma**
A comma was missing.

**Missing numeric value**
The number was missing after a numeric capability.

**NULL string value**
The proper way to say that a string capability does not exist is to cancel it.

**Very long string found. Missing comma?**
Self-explanatory.

**Unknown option. Usage is:**
An invalid option was entered.

**Too many file names. Usage is:**
Self-explanatory.

*name* **non-existent or permission denied**
The given directory could not be written into.

*name* **is not a directory**
Self-explanatory.

*name*: **Permission denied**
Access denied.

*name*: **Not a directory**
**tic** wanted to use the given name as a directory, but it already exists as a file

**SYSTEM ERROR!! Fork failed!!!**
A **fork**(2) failed.

**Error in following up use-links.**
Either there is a loop in the links or they reference non-existent terminals. The following is a list of the entries involved:
A **terminfo**(5V) entry with a **use**=*name* capability either referenced a non-existent terminal called *filename* or *filename* somehow referred back to the given entry.

NAME

timed – DARPA Time server

SYNOPSIS

**/usr/etc/in.timed**

DESCRIPTION

**timed** is a server which supports the DARPA Time Server Protocol. The time server operates at the port indicated in the "time" service description; see **services**(5), and is invoked by **inetd**(8C) each time there is a connection to the time server.

SEE ALSO

**services**(5), **rdate**(8), **inetd**(8C)

BUGS

A more sophisticated facility that can accept broadcasts and synchronize clocks over an internet is needed.

## NAME

tnamed – DARPA Trivial name server

## SYNOPSIS

**/usr/etc/in.tnamed** [ –v ]

## DESCRIPTION

**tnamed** is a server that supports the DARPA Name Server Protocol. The name server operates at the port indicated in the "name" service description (see **services**(5)), and is invoked by **inetd**(8C) when a request is made to the name server.

Two known clients of this service are the MIT PC/IP software the Bridge boxes.

## OPTIONS

–v Invoke the daemon in verbose mode.

## SEE ALSO

**uucp**(1C), **services**(5), **inetd**(8C)

Postel, Jon, *Internet Name Server*, IEN 116, SRI International, Menlo Park, California, August 1979.

## BUGS

The protocol implemented by this program is obsolete. Its use should be phased out in favor of the Internet Domain protocol. See **named**(8C).

NAME

trpt – transliterate protocol trace

SYNOPSIS

**/usr/etc/trpt** [ **–afjst** ] [ **–p***hex-address* ] [ *system* [ *core* ] ]

DESCRIPTION

**trpt** interrogates the buffer of TCP trace records created when a socket is marked for "debugging" (see **getsockopt**(2)), and prints a readable description of these records. When no options are supplied, **trpt** prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

OPTIONS

**–a** In addition to the normal output, print the values of the source and destination addresses for each packet recorded.

**–f** Follow the trace as it occurs, waiting a short time for additional records each time the end of the log is reached.

**–j** Just give a list of the protocol control block addresses for which there are trace records.

**–s** In addition to the normal output, print a detailed description of the packet sequencing information.

**–t** In addition to the normal output, print the values for all timers at each point in the trace.

**–p** hex-address
Show only trace records associated with the protocol control block, the address of which follows.

The recommended use of **trpt** is as follows. Isolate the problem and enable debugging on the **socket**(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **–A** option to **netstat**(8C). Then run **trpt** with the **–p** option, supplying the associated protocol control block addresses. The **–f** option can be used to follow the trace log once the trace is located. If there are many sockets using the debugging option, the **–j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

**/vmunix**
**/dev/kmem**

SEE ALSO

**getsockopt**(2), **netstat**(8C)

DIAGNOSTICS

**no namelist**
When the system image does not contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the trace record.

The output format is inscrutable and should be described here.

## NAME

tunefs – tune up an existing file system

## SYNOPSIS

**/usr/etc/tunefs** [ **–a** *maxcontig* ] [ **–d** *rotdelay* ] [ **–e** *maxbpg* ] [ **–m** *minfree* ] *special* | *filesystem*

## DESCRIPTION

**tunefs** is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the OPTIONS given below:

## OPTIONS

**–a** *maxcontig*

This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see **–d** below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

**–d** *rotdelay*

This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

**–e** *maxbpg*

This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

**–m** *minfree*

This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note: if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

## SEE ALSO

**fs**(5), **dumpfs**(8), **mkfs**(8), **newfs**(8)

*System and Network Administration*

## BUGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems; if run on the root file system, the system must be rebooted.

NAME

tzsetup – set up old-style time zone information in the kernel

SYNOPSIS

**/usr/etc/tzsetup**

DESCRIPTION

**tzsetup** attempts to find the offset from GMT and old-style Daylight Savings Time correction type (see **gettimeofday**(2)) that most closely matches the default time zone for the machine, and to pass this information to the kernel with a **settimeofday** ( ) call (see **gettimeofday**(2)). This is necessary if programs built under releases of SunOS prior to 4.0 are to be run; those programs get time zone information from the kernel using **gettimeofday.**

If it cannot find the offset from GMT, the offset is set to 0; if it cannot find the Daylight Savings Time correction type, it is set to DST_NONE, indicating that no Daylight Savings Time correction is to be performed.

DIAGNOSTICS

**tzsetup: Can't open /usr/share/lib/zoneinfo/localtime:** *reason*
The time zone file for the current time zone could not be opened.

**tzsetup: Error reading /usr/lib/zoneinfo/localtime:** *reason*
The time zone file for the current time zone could not be read.

**tzsetup: Two or more time zone types are equally valid — no DST selected**
There were two or more Daylight Savings Time correction types that generated results that were equally close to the correct results. None of them was selected. Programs built under versions of SunOS prior to 4.0 may not convert dates correctly.

**tzsetup: No old-style time zone type is valid — no DST selected**
None of the Daylight Savings Time correction types generated results that were in any way correct; none of them was selected. Programs built under versions of SunOS prior to 4.0 may not convert dates correctly.

**tzsetup: Warning: No old-style time zone type is completely valid**
None of the Daylight Savings Time correction types generated results that were completely correct; the best of them was selected. Programs built under versions of SunOS prior to 4.0 may not convert dates correctly.

**tzsetup: Can't set time zone**
**tzsetup** was run by a user other than the super-user; only the super-user may change the kernel's notion of the current time zone.

SEE ALSO

**gettimeofday**(2), **tzfile**(5), **zic**(8)

## NAME

unconfigure – reset the network configuration for a Sun386i system

## SYNOPSIS

**/usr/etc/unconfigure** [ **–y** ]

## AVAILABILITY

Sun386i systems only.

## DESCRIPTION

**unconfigure** restores most of the system configuration and status files to the state they were in when delivered by Sun Microsystems, Inc. It also deletes all user accounts (including home directories), Yellow Pages information, and any diskless client configurations that were set up.

After running **unconfigure,** a system halts. Rebooting it to multi-user mode at this point will start automatic system installation.

**unconfigure** is intended for use in the following situations:

- As one of the final steps in Software Manufacturing.
- In systems being set up with temporary configurations, holding no user accounts or diskless clients. These will occur during demonstrations and evaluation trials.
- To allow systems that had been used as standalones to be upgraded to join a network in a role other than as a master server. (See instructions later.)

**unconfigure** is potentially a dangerous utility; it does not work unless invoked by the super-user. As a warning, unless the **–y** option is passed, it will require confirmation that all user files and system software configuration information is to be deleted.

This utility is *not* recommended for routine use of any sort.

### Resetting Temporary Configurations

If users need to set up and tear down configurations, **unconfigure** can be used to restore the system to an essentially as-manufactured state. The main concern here is that user accounts will be deleted, so this should not be done casually.

To reset a temporary configuration, just become the super-user and invoke **unconfigure.**

### Upgrading Standalones to Network Clients

Systems that are going to be networked should be networked from the very first, if at all possible. This eliminates whole classes of compatibility problems, such as pathnames and (in particular) user account clashes.

Automatic system installation directly supports upgrading a single standalone system to a YP master, and joining any number of unused systems (or systems upon which **unconfigure** has been run) into a network.

However, in the situation where standalone systems that have been used extensively are to be joined to a network, **unconfigure** can be used in conjunction with automatic system installation by a knowledgeable super-user to change a system's configuration from standalone to network client. This procedure is not recommended for use by inexperienced administrators.

The following procedure is not needed unless user accounts or other data need to be preserved; it is intended to ensure that every UID and GID is changed so as not to clash with those in use on the network. It must be applied to each system that is being upgraded from a standalone to a network client.

The procedure is as follows:

1. Identify all accounts and files that you'll want to save. If there are none, just run **unconfigure** and install the system on the network. Do not follow the remaining steps.
2. Copy **/etc/passwd** to **/etc/passwd.bak.**
3. Rename all the files (including home directories) so that they aren't deleted. (See FILES below.) These will probably be only in **/export/home.**

4. Run **unconfigure** and install the system on the network.
5. For each account listed in **/etc/passwd.bak** that you want to save, follow this procedure:
   a. Create a new account on the network; if the UID and GID are the same as in **/etc/passwd.bak** on the standalone, then skip the next step. However, be sure that you do not make two different accounts with the same UID.
   b. Use the '**chown –R**' command to change the ownership of the home directories.
   c. You may need to rename the files you just chowned above, for example to ensure that they are the user's home directory. This may involve updating the **auto.home**(5) and **auto.home**(5) YP maps, as well.
   6. Delete **/etc/passwd.bak**.

FILES

**unconfigure** deletes the following files, if they are present, replacing some of them with the distribution version if one is supposed to exist:

| | | | |
|---|---|---|---|
| **/etc/.rootkey** | **/etc/ethers** | **/etc/localtime** | **/etc/publickey** |
| **/etc/auto.home** | **/etc/exports** | **/etc/net.conf** | **/etc/sendmail.cf** |
| **/etc/auto.vol** | **/etc/fstab** | **/etc/netmasks** | **/etc/syslog.conf** |
| **/etc/bootparams** | **/etc/group** | **/etc/networks** | **/etc/systems** |
| **/etc/bootservers** | **/etc/hosts** | **/etc/passwd** | **/single/ifconfig** |
| **/var/sysex/*** | | | |

and all files in **/var/yp** except those distributed with the operating system.

**unconfigure** truncates all files in **/var/adm**. All user home directories in **/export/home** are deleted, except those for the default user account **users**, which is shipped with the operating system. All diskless client configuration information stored in **/export/roots**, **/export/swaps**, and **/export/dumps** is deleted.

SEE ALSO

**find**(1), **passwd**(5), **group**(5), **adduser**(8), **chgrp**(1), **chown**(8)

BUGS

More of the system configuration files should be reset.

This does not yet support taking a workstation off the network temporarily, for example, to take it home over the weekend for use as a standalone, or to move it to another network while travelling. This should be the default behavior.

The procedure for upgrading standalones to network clients should be automated; currently, only upgrading a standalone to a master server is automated.

NAME

update – periodically update the super block

SYNOPSIS

**/usr/etc/update**

DESCRIPTION

**update** is a program that executes the **sync**(2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO

**sync**(2), **init**(8), **sync**(8)

## NAME

uuclean – uucp spool directory clean-up

## SYNOPSIS

**/usr/lib/uucp/uuclean** [ **–m** ] [ **–n***time* ] [ **–p***pre* ]

## DESCRIPTION

**uuclean** scans the spool directory for files with the specified prefix and deletes all those which are older than the specified number of hours.

## OPTIONS

**–m** Send mail to the owner of the file when it is deleted.

**–n***time* Files whose age is more than *time* hours are deleted if the prefix test is satisfied (default time is 72 hours).

**–p***pre* Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following deletes all files older than the specified time.

**uuclean** will typically be started by **cron**(8).

## FILES

| | |
|---|---|
| **/usr/lib/uucp** | directory with commands used by uuclean internally |
| **/usr/lib/uucp/spool** | spool directory |

## SEE ALSO

**uucp**(1C), **uux**(1C), **cron**(8)

NAME

vipw – edit the password file

SYNOPSIS

**/usr/etc/vipw**

DESCRIPTION

**vipw** edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The **vi**(1) editor will be used unless the environment variable **VISUAL** or **EDITOR** indicates an alternate editor.

**vipw** performs a number of consistency checks on the password entry for root, and will not allow a password file with a "mangled" root entry to be installed. It also checks the **/etc/shells** file to verify the login shell for root.

FILES

**/etc/ptmp**
**/etc/shells**

SEE ALSO

**passwd**(1), **vi**(1), **passwd**(5), **adduser**(8)

**NAME**

vmstat – report virtual memory statistics

**SYNOPSIS**

**vmstat** [ **–fisS** ] [ *interval* [ *count* ] ]

**DESCRIPTION**

**vmstat** delves into the system and normally reports certain statistics kept about process, virtual memory, disk, trap and CPU activity.

Without options, **vmstat** displays a one-line summary of the virtual memory activity since the system has been booted. If *interval* is specified, **vmstat** summarizes activity over the last *interval* seconds. If a *count* is given, the statistics are repeated *count* times.

For example, the following command displays a summary of what the system is doing every five seconds. This is a good choice of printing interval since this is how often some of the statistics are sampled in the system.

```
example% vmstat 5

procs   memory              page    faults
r b w   avm  fre  re at  pi po fr  de sr x0 x1 x2 x3  in  sy  cs us sy id
2 0 0  918  286  0  0   0  0  0   0  0  1  0  0  0   4  12   5  3  5 91
1 0 0  846  254  0  0   0  0  0   0  0  6  0  1  0  42 153  31  7 40 54
1 0 0  840  268  0  0   0  0  0   0  0  5  0  0  0  27 103  25  8 26 66
1 0 0  620  312  0  0   0  0  0   0  0  6  0  0  0  26  76  25  6 27 67

^C
example%
```

The fields of **vmstat**'s display are:

**procs** Report the number of processes in each of the three following states:

- **r** in run queue
- **b** blocked for resources (i/o, paging, etc.)
- **w** runnable or short sleeper (< 20 secs) but swapped

**memory** Report on usage of virtual and real memory. Virtual memory is considered active if it belongs to processes which are running or have run in the last 20 seconds.

- **avm** number of active virtual Kbytes
- **fre** size of the free list in Kbytes

**page** Report information about page faults and paging activity. The information on each of the following activities is averaged each five seconds, and given in units per second.

- **re** page reclaims — but see the –S option for how this field is modified.
- **at** number of attaches — but see the –S option for how this field is modified.
- **pi** kilobytes per second paged in
- **po** kilobytes per second paged out
- **fr** kilobytes freed per second
- **de** anticipated short term memory shortfall in Kbytes
- **sr** pages scanned by clock algorithm, per-second

**disk** Report number of disk operations per second (this field is system dependent). For Sun systems, four slots are available for up to four drives: "x0" (or "s0" for SCSI disks), "x1", "x2", and "x3".

**faults** Report trap/interrupt rate averages per second over last 5 seconds.

- **in** (non clock) device interrupts per second
- **sy** system calls per second
- **cs** CPU context switch rate (switches/sec)

**cpu** Give a breakdown of percentage usage of CPU time.
- **us** user time for normal and low priority processes
- **sy** system time
- **id** CPU idle

OPTIONS

**–f** Report on the number of **forks** and **vforks** since system startup and the number of pages of virtual memory involved in each kind of fork.

**–i** Report the number of interrupts per device. Autovectored interrupts (including the clock) are listed first.

**–s** Display the contents of the **sum** structure, giving the total number of several kinds of paging-related events which have occurred since boot.

**–S** Report on swapping rather than paging activity. This option will change two fields in **vmstat**'s "paging" display: rather than the "re" and "at" fields, **vmstat** will report "si" (swap-ins), and "so" (swap-outs).

FILES

**/dev/kmem**
**/vmunix**

BUGS

If more than one autovectored device has the same name, interrupts are counted for all like-named devices regardless of unit number. Such devices are listed with a unit number of '**?**'.

NAME

ypinit - build and install Yellow Pages database

SYNOPSIS

**/usr/etc/yp/ypinit –m**

**/usr/etc/yp/ypinit –s** *master_name*

DESCRIPTION

**ypinit** sets up a Yellow Pages database on a YP server. It can be used to set up a master or a slave server. You must be the super-user to run it. It asks a few, self-explanatory questions, and reports success or failure to the terminal.

It sets up a master server using the simple model in which that server is master to all maps in the data base. This is the way to bootstrap the YP system; later if you want you can change the association of maps to masters. All databases are built from scratch, either from information available to the program at runtime, or from the ASCII data base files in **/etc**. These files are listed below under FILES. All such files should be in their "traditional" form, rather than the abbreviated form used on client machines.

A YP database on a slave server is set up by copying an existing database from a running server. The *master_name* argument should be the hostname of YP server (either the master server for all the maps, or a server on which the data base is up-to-date and stable).

Read **ypfiles**(5) and **ypserv**(8) for an overview of the Yellow Pages.

OPTIONS

**–m** Indicate that the local host is to be the YP master.

**–s** Set up a slave database.

FILES

**/etc/passwd**
**/etc/group**
**/etc/hosts**
**/etc/networks**
**/etc/services**
**/etc/protocols**
**/etc/ethers**

SEE ALSO

**makedbm**(8), **ypfiles**(5), **ypmake**(8), **yppush**(8), **ypserv**(8), **ypxfr**(8)

NAME

ypmake – rebuild Yellow Pages database

SYNOPSIS

**cd /var/yp ; make** [ *map* ]

DESCRIPTION

The file called **Makefile** in **/var/yp** is used by **make** to build the Yellow Pages database. With no arguments, **make** creates **dbm** databases for any YP maps that are out-of-date, and then executes **yppush**(8) to notify slave databases that there has been a change.

If you supply a *map* on the command line, **make** will update that map only. Typing **make passwd** will create and **yppush** the password database (assuming it is out of date). Likewise, **make hosts** and **make networks** will create and **yppush** the host and network files, **/etc/hosts** and **/etc/networks**.

There are three special variables used by **make**: DIR, which gives the directory of the source files; NOPUSH, which when non-null inhibits doing a **yppush** of the new database files; and DOM, used to construct a domain other than the master's default domain. The default for DIR is **/etc**, and the default for NOPUSH is the null string.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the YP.

FILES

**/var/yp**
**/etc/hosts**
**/etc/networks**

SEE ALSO

**make**(1), **ypfiles**(5), **makedbm**(8), **yppush**(8), **ypserv**(8)

## NAME

yppasswdd – server for modifying Yellow Pages password file

## SYNOPSIS

**/usr/etc/rpc.yppasswdd** *filename* [ *adjunct_file* ] [ **–m** *argument1 argument2 ...* ]

## AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

## DESCRIPTION

**yppasswdd** is a server that handles password change requests from **yppasswd**(1). Unless an *adjunct_file* is specified, it changes a password entry in *filename*, which is assumed to be in the format of **passwd**(5). If an *adjunct_file* is specified or **/etc/security/passwd.adjunct** exists, this file will be changed instead of the *filename*. An entry in *filename* or *adjunct_file* will only be changed if the password presented by **yppasswd**(1) matches the encrypted password of that entry.

If the **–m** option is given, then after *filename* or *adjunct_file* is modified, a **make**(1) will be performed in */var/yp*. Any arguments following the flag will be passed to *make*.

This server is not run by default, nor can it be started up from **inetd**(8C). If it is desired to enable remote password updating for the Yellow Pages, then an entry for **yppasswdd** should be put in the **/etc/rc** file of the host serving as the master for the Yellow Pages **passwd** file.

## EXAMPLE

If the Yellow Pages password file is stored as **/var/yp/src/passwd**, then to have password changes propagated immediately, the server should be invoked as

**/usr/etc/rpc.yppasswdd /var/yp/src/passwd –m passwd DIR=/var/yp/src**

In this case, **src** is the YP domain name.

## FILES

**/var/yp/Makefile**
**/etc/security/passwd.adjunct**
**/etc/rc**

## SEE ALSO

**make**(1), **yppasswd**(1), **passwd**(5), **passwd.adjunct**(5), **ypfiles**(5), **inetd**(8C), **ypmake**(8)

NAME

yppoll - what version of a YP map is at a YP server host

SYNOPSIS

**/usr/etc/yp/yppoll** [ **-h** *host* ] [ **-d** *domain* ] *mapname*

DESCRIPTION

**yppoll** asks a **ypserv**(8) process what the order number is, and which host is the master YP server for the named map. If the server is a v.1 YP protocol server, **yppoll** uses the older protocol to communicate with it. In this case, it also uses the older diagnostic messages in case of failure.

OPTIONS

**-h** *host* Ask the **ypserv** process at *host* about the map parameters. If *host* is not specified, the YP server for the local host is used. That is, the default host is the one returned by **ypwhich**(8).

**-d** *domain*

Use *domain* instead of the default domain.

SEE ALSO

**ypfiles**(5), **ypserv**(8), **ypwhich**(8)

## NAME

yppush - force propagation of a changed YP map

## SYNOPSIS

**/usr/etc/yp/yppush** [ **–v** ] [ **–d** *domain* ] *mapname*

## DESCRIPTION

**yppush** copies a new version of a Yellow Pages (YP) map from the master YP server to the slave YP servers. It is normally run only on the master YP server by the **Makefile** in **/var/yp** after the master databases are changed. It first constructs a list of YP server hosts by reading the YP map **ypservers** within the *domain*. Keys within the map **ypservers** are the ASCII names of the machines on which the YP servers run.

A "transfer map" request is sent to the YP server at each host, along with the information needed by the transfer agent (the program which actually moves the map) to call back the **yppush** . When the attempt has completed (successfully or not), and the transfer agent has sent **yppush** a status message, the results may be printed to stdout. Messages are also printed when a transfer is not possible; for instance when the request message is undeliverable, or when the timeout period on responses has expired.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the Yellow Pages.

## OPTIONS

**–d** Specify a *domain*.

–v Verbose. This causes messages to be printed when each server is called, and for each response. If this flag is omitted, only error messages are printed.

## FILES

**/var/yp/***domain***/ypservers.{dir,***pag***}**
**/var/yp**

## SEE ALSO

**ypfiles**(5), **ypserv**(8), **ypxfr**(8), YP protocol specification

## BUGS

In the current implementation (version 2 YP protocol), the transfer agent is **ypxfr**(8), which is started by the **ypserv** program. If **yppush** detects that it is speaking to a version 1 YP protocol server, it uses the older protocol, sending a version 1 YPPROC_GET request and issues a message to that effect. Unfortunately, there is no way of knowing if or when the map transfer is performed for version 1 servers. **yppush** prints a message saying that an "old-style" message has been sent. The system administrator should later check to see that the transfer has actually taken place.

NAME

ypserv, ypbind – Yellow Pages server and binder processes

SYNOPSIS

**/usr/etc/ypserv**

**/usr/etc/ypbind**

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

The Yellow Pages (YP) provides a simple network lookup service consisting of databases and processes. The databases are **dbm**(3X) files in a directory tree rooted at **/var/yp**. These files are described in **ypfiles**(5). The processes are **/usr/etc/ypserv**, the YP database lookup server, and **/usr/etc/ypbind**, the YP binder. The programmatic interface to YP is described in **ypclnt**(3N). Administrative tools are described in **yppush**(8), **ypxfr**(8), **yppoll**(8), **ypwhich**(8), and **ypset**(8). Tools to see the contents of YP maps are described in **ypcat**(1), and **ypmatch**(1). Database generation and maintenance tools are described in **ypinit**(8), **ypmake**(8), and **makedbm**(8).

Both **ypserv** and **ypbind** are daemon processes typically activated at system startup time from **/etc/rc.local**. **ypserv** runs only on YP server machines with a complete YP database. **ypbind** runs on all machines using YP services, both YP servers and clients.

The **ypserv** daemon's primary function is to look up information in its local database of YP maps. The operations performed by **ypserv** are defined for the implementor by the *YP Protocol Specification*, and for the programmer by the header file **<rpcsvc/yp_prot.h>**. Communication to and from **ypserv** is by means of RPC calls. Lookup functions are described in **ypclnt**(3N), and are supplied as C-callable functions in the C library. There are four lookup functions, all of which are performed on a specified map within some YP domain: *Match*, *Get_first*, *Get_next*, and *Get_all*. The *Match* operation takes a key, and returns the associated value. The *Get_first* operation returns the first key-value pair from the map, and *Get_next* can be used to enumerate the remainder. *Get_all* ships the entire map to the requester as the response to a single RPC request.

Two other functions supply information about the map, rather than map entries: *Get_order_number*, and *Get_master_name*. In fact, both order number and master name exist in the map as key-value pairs, but the server will not return either through the normal lookup functions. (If you examine the map with **makedbm**(8), however, they will be visible.) Other functions are used within the YP subsystem itself, and are not of general interest to YP clients. They include *Do_you_serve_this_domain?*, *Transfer_map*, and *Reinitialize_internal_state*.

The function of **ypbind** is to remember information that lets client processes on a single node communicate with some **ypserv** process. **ypbind** must run on every machine which has YP client processes; **ypserv** may or may not be running on the same node, but must be running somewhere on the network.

The information **ypbind** remembers is called a *binding* — the association of a domain name with the internet address of the YP server, and the port on that host at which the **ypserv** process is listening for service requests. This information is cached in the directory **/var/yp/binding** using a filename of **domainname.version**.

The process of binding is driven by client requests. As a request for an unbound domain comes in, the **ypbind** process broadcasts on the net trying to find a **ypserv** process that serves maps within that domain. Since the binding is established by broadcasting, there must be at least one **ypserv** process on every net. If the client is running in C2 secure mode, then **ypbind** will only accept bindings to servers where the **ypserv** process is running as root. Once a domain is bound by a particular **ypbind**, that same binding is given to every client process on the node. The **ypbind** process on the local node or a remote node may be queried for the binding of a particular domain by using the **ypwhich**(1) command.

Bindings and rebindings are handled transparently by the C library routines. If **ypbind** is unable to speak to the **ypserv** process it's bound to, it marks the domain as unbound, tells the client process that the domain is unbound, and tries to bind the domain once again. Requests received for an unbound domain will wait until the domain requested is bound. In general, a bound domain is marked as unbound when the node running **ypserv** crashes or gets overloaded. In such a case, **ypbind** will to bind any YP server (typically one that is less-heavily loaded) available on the net.

**ypbind** also accepts requests to set its binding for a particular domain. The request is usually generated by the YP subsystem itself. **ypset**(8) is a command to access the *Set_domain* facility. It is for unsnarling messes. Note that the *Set Domain* procedure only accepts requests from processes running as root.

FILES

If the file **/var/yp/ypserv.log** exists when **ypserv** starts up, log information will be written to this file when error conditions arise.

The file(s) **/var/yp/binding/domainname.version** will be created to speed up the binding process. These files cache the last successful binding created for the given domain, when a binding is requested these files are checked for validity and then used.
**/var/yp**
**/usr/etc/ypbind**

SEE ALSO

**domainname**(1), **ypcat**(1), **ypmatch**(1), **dbm**(3X), **ypclnt**(3N), **ypfiles**(5), **makedbm**(8), **ypmake**(8), **ypinit**(8), **yppoll**(8), **yppush**(8), **ypset**(8), **ypwhich**(8), **ypxfr**(8)

*YP Protocol Specification* in *Network Programming*

NOTES

Both **ypbind** and **ypserv** support multiple domains. The **ypserv** process determines the domains it serves by looking for directories of the same name in the directory **/var/yp**. It will reply to all broadcasts requesting yp service for that domain. Additionally, the **ypbind** process can maintain bindings to several domains and their servers, the default domain is however the one specified by the **domainname**(1) command at startup time.

NAME

ypset - point ypbind at a particular server

SYNOPSIS

**/usr/etc/yp/ypset** [ **–V1** | **–V2** ] [ **–d** *domain* ] [ **–h** *host* ] *server*

DESCRIPTION

**ypset** tells **ypbind** to get YP services for the specified *domain* from the **ypserv** process running on *server*. If *server* is down, or isn't running **ypserv**, this is not discovered until a YP client process tries to get a binding for the domain. At this point, the binding set by **ypset** will be tested by **ypbind**. If the binding is invalid, **ypbind** will attempt to rebind for the same domain.

**ypset** is useful for binding a client node which is not on a broadcast net, or is on a broadcast net which isn't running a YP server host. It also is useful for debugging YP client applications, for instance where a YP map only exists at a single YP server host.

In cases where several hosts on the local net are supplying YP services, it is possible for **ypbind** to rebind to another host even while you attempt to find out if the **ypset** operation succeeded. For example, you can type:

**example% ypset host1**
**example% ypwhich**
**host2**

which can be confusing. This is a function of the YP subsystem's attempt to load-balance among the available YP servers, and occurs when *host1* does not respond to **ypbind** because it is not running **ypserv** (or is overloaded), and *host2*, running **ypserv**, gets the binding.

*server* indicates the YP server to bind to, and can be specified as a name or an IP address. If specified as a name, **ypset** will attempt to use YP services to resolve the name to an IP address. This will work only if the node has a current valid binding for the domain in question. In most cases, *server* should be specified as an IP address.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the Yellow Pages.

OPTIONS

**–V1** Bind *server* for the (old) v.1 YP protocol.

**–V2** Bind *server* for the (current) v.2 YP protocol.

If no version is supplied, **ypset**, first attempts to set the domain for the (current) v.2 protocol. If this attempt fails, **ypset**, then attempts to set the domain for the (old) v.1 protocol.

**–h***host* Set ypbind's binding on *host*, instead of locally. *host* can be specified as a name or as an IP address.

**–d***domain*

Use *domain* , instead of the default domain.

SEE ALSO

**ypwhich**(1), **ypfiles**(5), **ypserv**(8)

## NAME

ypupdated – server for changing YP information

## SYNOPSIS

**rpc.ypupdated** [ **–is** ]

## DESCRIPTION

**ypupdated** is a daemon that updates information in the Yellow Pages, normally started up by **inetd**(8C). **ypupdated** consults the file **updaters**(5) in the directory **/var/yp** to determine which YP maps should be updated and how to change them.

By default, the daemon requires the most secure method of authentication available to it, either DES (secure) or UNIX (insecure).

## OPTIONS

**–s** accept only calls authenticated using the secure RPC mechanism (AUTH_DES authentication). This disables programmatic updating of YP maps unless the network supports these calls.

**–i** also accept RPC calls with the insecure AUTH_UNIX credentials. This allows programmatic updating of YP maps in all networks.

## FILES

**/var/yp/updaters**

## SEE ALSO

**updaters**(5), **inetd(8C)**, **keyserv**(8C)

*System and Network Administration*

*Network Programming*

NAME

ypwhich – what machine is the YP server?

SYNOPSIS

**ypwhich** [ **–d** *domainname* ] [ *hostname* ]

**ypwhich** [ **–d** *domainname* ] [ **–t** ] **–m** [ *mname* ]

AVAILABILITY

This program is available with the *Networking Tools and Programs* software installation option. Refer to *Installing the SunOS* for information on how to install optional software.

DESCRIPTION

**ypwhich** tells which YP server supplies Yellow Pages to a YP client, and which YP server is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *hostname* is specified, that machine is queried to find out which YP master it is using.

If the **–m** switch is used without *mname*, a list of every map in the domain and the master of each will be printed. If *mname* is specified, only the master YP server for that map is printed. *mname* may be a mapname, or a nickname for a mapname. Mapnames and nicknames are described in **ypcat**(1).

OPTIONS

**–d** *Domainname* specifies the name of a YP domain. The default is the default domain for the local machine.

**–m** Find the master YP server for a map, or for all maps in a domain. No *hostname* may be specified with **–m**.

**–t** Inhibit nickname translation; useful if there is a mapname identical to a nickname. This is not true of any Sun-supplied map.

SEE ALSO

**ypcat**(1), **ypfiles**(5), **rpcinfo**(8C), **yppush**(8), **ypserv**(8)

## NAME

ypxfr – transfer YP map from a YP server to here

## SYNOPSIS

**/usr/etc/yp/ypxfr** [ **–f** ] [ **–c** ] [ **–d** *domain* ] [ **–h** *host* ] [ **–s** *domain* ] [ **–C** *tid prog ipadd port* ] *mapname*

## DESCRIPTION

**ypxfr** moves a YP map in the default domain for the local host to the local host by making use of normal YP services. It creates a temporary map in the directory **/var/yp/***domain* (this directory must already exist; *domain* is the default domain for the local host), fills it by enumerating the map's entries, fetches the map parameters (master and order number), and loads them. It then deletes any old versions of the map and moves the temporary map to the real *mapname*.

If run interactively, **ypxfr** it writes its output to the terminal. However, if it is invoked without a controlling terminal, and if the log file **/var/yp/ypxfr.log** exists, it will append all its output to that file. Since **ypxfr** is most often run from the super-user's **crontab** file, or by **ypserv**, you can use the log file to retain a record of what was attempted, and what the results were.

If **issecure**(3) is TRUE, **ypxfr** requires that **ypserv** on the *host* be running as root. If the map being transferred is a secure map, **ypxfr** sets the permissions on the map to 0600.

For consistency between servers, **ypxfr** should be run periodically for every map in the YP data base. Different maps change at different rates: the *services.byname* map may not change for months at a time, for instance, and may therefore be checked only once a day (in the wee hours). You may know that *mail.aliases* or *hosts.byname* changes several times per day. In such a case, you may want to check hourly for updates. A **crontab**(5) entry can be used to perform periodic updates automatically. Rather than having a separate **crontab** entry for each map, you can group comands to update several maps in a shell script. Examples (mnemonically named) are in **/usr/etc/yp**: **ypxfr_1perday**, **ypxfr_2perday**, and **ypxfr_1perhour**. They can serve as reasonable first cuts.

Refer to **ypfiles**(5) and **ypserv**(8) for an overview of the Yellow Pages.

## OPTIONS

**–f** Force the transfer to occur even if the version at the master is not more recent than the local version.

**–c** Do not send a "Clear current map" request to the local **ypserv** process. Use this flag if **ypserv** is not running locally at the time you are running **ypxfr**. Otherwise, **ypxfr** will complain that it can't talk to the local **ypserv**, and the transfer will fail.

**–d***domain*
Specify a domain other than the default domain.

**–h***host* Get the map from *host*, regardless of what the map says the master is. If *host* is not specified, **ypxfr** will ask the YP service for the name of the master, and try to get the map from there. *host* may be a name or an internet address in the form *a.b.c.d*.

**–s***domain*
Specify a source domain from which to transfer a map that should be the same across domains (such as the *services.byname* map).

**–C***tid prog ipadd port*
This option is **only** for use by **ypserv**. When **ypserv** invokes **ypxfr**, it specifies that **ypxfr** should call back a **yppush** process at the host with IP address *ipaddr*, registered as program number *prog*, listening on port *port*, and waiting for a response to transaction *tid*.

## FILES

**/var/yp/ypxfr.log** log file
**/usr/etc/yp/ypxfr_1perday**
script to run one transfer per day, for use with **cron**(8)

**/usr/etc/yp/ypxfr_2perday**
script to run two transfers per day
**/usr/etc/yp/ypxfr_1perhour**
script for hourly transfers of volatile maps
**/var/yp/***domain* YP domain
**/var/spool/cron/crontabs/root**
Super-user's **crontab** file

**SEE ALSO**

**issecure**(3), **crontab**(5), **ypfiles**(5), **cron**(8), **ypserv**(8), **yppush**(8),
*YP Protocol Specification*, in *Network Programming*

**NAME**

zdump – time zone dumper

**SYNOPSIS**

**zdump** [ **–v** ] [ **–c** *cutoffyear* ] [ *zonename* ... ]

**DESCRIPTION**

**zdump** prints the current time in each *zonename* named on the command line.

**OPTIONS**

**–v** For each *zonename* on the command line, print the current time, the time at the lowest possible time value, the time one day after the lowest possible time value, the times both one second before and exactly at each time at which the rules for computing local time change, the time at the highest possible time value, and the time at one day less than the highest possible time value. Each line ends with **isdst=1** if the given time is Daylight Saving Time or **isdst=0** otherwise.

**–c** *cutoffyear*

Cut off the verbose output near the start of the year *cutoffyear*.

**FILES**

**/usr/share/lib/zoneinfo** standard zone information directory

**SEE ALSO**

**ctime**(3), **tzfile**(5), **zic**(8)

NAME

zic – time zone compiler

SYNOPSIS

**zic** [ **–v** ] [ **–d** *directory* ] [ **–l** *localtime* ] [ *filename* ... ]

DESCRIPTION

**zic** reads text from the file(s) named on the command line and creates the time conversion information files specified in this input. If a *filename* is '–', the standard input is read.

Input lines are made up of fields. Fields are separated from one another by any number of white space characters. Leading and trailing white space on input lines is ignored. An '#' (unquoted sharp character) in the input introduces a comment which extends to the end of the line the sharp character appears on. White space characters and sharp characters may be enclosed in ' "' (double quotes) if they're to be used as part of a field. Any line that is blank (after comment stripping) is ignored. Non-blank lines are expected to be of one of three types: rule lines, zone lines, and link lines.

A rule line has the form

**Rule NAME FROM TO TYPE IN ON AT SAVE LETTER/S**

For example:

**Rule USA 1969 1973 – Apr lastSun 2:00 1:00 D**

The fields that make up a rule line are:

**NAME** Gives the (arbitrary) name of the set of rules this rule is part of.

**FROM** Gives the first year in which the rule applies. The word **minimum** (or an abbreviation) means the minimum year with a representable time value. The word **maximum** (or an abbreviation) means the maximum year with a representable time value.

**TO** Gives the final year in which the rule applies. In addition to **minimum** and **maximum** (as above), the word **only** (or an abbreviation) may be used to repeat the value of the **FROM** field.

**TYPE** Gives the type of year in which the rule applies. If **TYPE** is '–' then the rule applies in all years between **FROM** and **TO** inclusive; if **TYPE** is **uspres**, the rule applies in U.S. Presidential election years; if **TYPE** is **nonpres**, the rule applies in years other than U.S. Presidential election years. If **TYPE** is something else, then **zic** executes the command

**yearistype** *year type*

to check the type of a year: an exit status of zero is taken to mean that the year is of the given type; an exit status of one is taken to mean that the year is not of the given type.

**IN** Names the month in which the rule takes effect. Month names may be abbreviated.

**ON** Gives the day on which the rule takes effect. Recognized forms include:

| | |
|---|---|
| **5** | the fifth of the month |
| **lastSun** | the last Sunday in the month |
| **lastMon** | the last Monday in the month |
| **Sun>=8** | first Sunday on or after the eighth |
| **Sun<=25** | last Sunday on or before the 25th |

Names of days of the week may be abbreviated or spelled out in full. Note: there must be no spaces within the ON field.

**AT** Gives the time of day at which the rule takes effect. Recognized forms include:

**2** time in hours

**2:00** time in hours and minutes

**15:00**
24-hour format time (for times after noon)

**1:28:14**
time in hours, minutes, and seconds

Any of these forms may be followed by the letter w if the given time is local "wall clock" time or s if the given time is local "standard" time; in the absence of w or s, wall clock time is assumed.

**SAVE** Gives the amount of time to be added to local standard time when the rule is in effect. This field has the same format as the AT field (although, of course, the w and s suffixes are not used).

**LETTER/S**
Gives the "variable part" (for example, the "S" or "D" in "EST" or "EDT") of time zone abbreviations to be used when this rule is in effect. If this field is '–', the variable part is null.

A zone line has the form

**Zone NAME GMTOFF RULES/SAVE FORMAT [UNTIL]**

For example:

**Zone Australia/South–west 9:30 Aus CST 1987 Mar 15 2:00**

The fields that make up a zone line are:

**NAME** The name of the time zone. This is the name used in creating the time conversion information file for the zone.

**GMTOFF**
The amount of time to add to GMT to get standard time in this zone. This field has the same format as the AT and SAVE fields of rule lines; begin the field with a minus sign if time must be subtracted from GMT.

**RULES/SAVE**
The name of the rule(s) that apply in the time zone or, alternately, an amount of time to add to local standard time. If this field is '–' then standard time always applies in the time zone.

**FORMAT**
The format for time zone abbreviations in this time zone. The pair of characters %s is used to show where the "variable part" of the time zone abbreviation goes. **UNTIL** The time at which the GMT offset or the rule(s) change for a location. It is specified as a year, a month, a day, and a time of day. If this is specified, the time zone information is generated from the given GMT offset and rule change until the time specified.

The next line must be a "continuation" line; this has the same form as a zone line except that the string "Zone" and the name are omitted, as the continuation line will place information starting at the time specified as the UNTIL field in the previous line in the file used by the previous line. Continuation lines may contain an UNTIL field, just as zone lines do, indicating that the next line is a further continuation.

A link line has the form

**Link LINK-FROM LINK-TO**

For example:

**Link US/Eastern EST5EDT**

The **LINK-FROM** field should appear as the **NAME** field in some zone line; the **LINK-TO** field is used as an alternate name for that zone.

Except for continuation lines, lines may appear in any order in the input.

OPTIONS

**–v** Complain if a year that appears in a data file is outside the range of years representable by system time values (0:00:00 AM GMT, January 1, 1970, to 3:14:07 AM GMT, January 19, 2038).

**–d** *directory*
Create time conversion information files in the directory **directory** rather than in the standard directory **/usr/share/lib/zoneinfo.**

**–l** *timezone*
Use the time zone *timezone* as local time. **zic** will act as if the file contained a link line of the form

Link *timezone* localtime

FILES

**/usr/share/lib/zoneinfo** standard directory used for created files

SEE ALSO

**time**(1V), **ctime**(3), **tzfile**(5), **zdump**(8)

NOTE

For areas with more than two types of local time, you may need to use local standard time in the **AT** field of the earliest transition time's rule to ensure that the earliest transition time recorded in the compiled file is correct.

# Index

## B

## C

## D

## E

## F

## G

## H



## I

## J



## K

## L

## M

## N

## O

## P

## Q



## R

## S

## T

## U

## V

## W



## X

## Y



## Z

# Notes

# Notes

# Notes