ULTRIX-11™
Programmer's Manual Volume 1

Order No. AA-X344B-TC

digital equipment corporation, maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a Digital Equipment Corporation license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

The following are trademarks of Digital Equipment Corporation:

DEC	Edusystem	ULTRIX
DEC/CMS	IAS	ULTRIX-11
DEC/MMS	MASSBUS	ULTRIX-32
DECnet	MICRO/PDP-11	ULTRIX-32m
DECsystem-10	Micro/RSX	UNIBUS
DECsystem-20	PDP	VAX
DECUS	PDT	VMS
DECwriter	RSTS	VT
DIBOL	RSX	digital ™

The following command descriptions as set forth in this document are copyrighted material of Digital Equipment Corporation:

bufstat(1m)	osload(1m)
cda(1m)	rasize(1m)
cdc(1)	rx2fmt(1m)
chog(1)	sysgen(1m)
chroot(1)	ted(1)
csf(1m)	tss(1m)
lpset(1m)	usat(1)
memstat(1m)	zaptty(1m)

In accordance with the licenses granted to Digital Equipment Corporation by AT&T Bell Laboratories and the University of California at Berkeley pertaining to the software described herein, the following should be understood by the licensee, and any related documentation provided by the licensee to third parties, whether pursuant to an agreement with Digital Equipment Corporation permitting sublicensing of the software described herein or otherwise, must contain the provisions of this section as set forth below:

a Information herein is derived from copyrighted material as permitted under a license agreement with AT&T Bell Laboratories.
Copyrighted © 1979 AT&T Bell Laboratories.

b UNIX is a trademark of AT&T Bell Laboratories.
The UNIX trademark may not be used in the name of any licensee's product. Any use of the trademark in advertising, publicity, packaging, labelling or otherwise must state that UNIX is a trademark of AT&T Bell Laboratories.

c This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following individuals and institutions for their role in its development:

The Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California. Ken Arnold, Earl T. Cohen, John Foderaro, Charles Haley, Mark Horton, William Joy, Jim Kleckner, Geoffrey Peck, Cliff Matthews, University of New Mexico; Eric Shienbrood.

"The UNIX Time-Sharing System": Copyright © 1974, Association for Computing Machinery, Inc. reprinted by permission. This is a revised version of an article that appeared in Communications of the ACM, 17, No. 7 (July 1974), pp.365-375. That article was a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15-17, 1973. Acknowledgements: for their help and support, R.H. Canaday, R. Morris, M.D. McIlroy, and J.F. Ossanna.

"Advanced Editing on UNIX" acknowledgement: Ted Dolotta for his ideas and assistance.

"Ex Reference Manual" acknowledgements: Chuck Haley contributed greatly to the early development of ex. Bruce Englar encouraged the redesign which led to ex version 1. Bill Joy wrote versions 1 and 2.0 through 2.7, and created the framework that users see in the present editor. Mark Horton added macros and other features and made the editor work on a large number of terminals and UNIX systems.

"An Introduction to the UNIX Shell" acknowledgements: Dennis Ritchie, John Mashey and Joe Maranzano for their help and support.

"An Introduction to the C Shell" acknowledgements: Michael Ubell, Eric Allman, Mike O'Brien, and Jim Kulp.

"LEARN – Computer-Aided Instruction on UNIX" acknowledgements: for their help and support, M.E. Bittrich, J.L. Blue, S.I. Feldman, P.A. Fox, M.J. McAlpin, E.Z. Rothkopf, Don Jackowski, and Tom Plum.

"A System for Typesetting Mathematics" acknowledgements: J.F. Ossanna, A.V. Aho, and S.C. Johnson, for their ideas and assistance.

"A TROFF Tutorial" acknowledgements: J.F. Ossanna, Jim Blinn, Ted Dolotta, Doug McIlroy, Mike Lesk and Joel Sturman, for their help and support.

The document "The C Programming Language – Reference Manual" is reprinted, with minor changes, from "The C Programming Language", by Brian W. Kernighan and Dennis M. Ritchie, Prentice-Hall, Inc., 1978.

"Make – A Program for Maintaining Computer Programs" acknowledgements: S.C. Johnson, and H. Gajewska, for their ideas and assistance.

"Screen Updating and Cursor Movement Optimization: A Library Package" acknowledgements: For their help and support, Bill Joy, Doug Merritt, Kurt Shoens, Ken Abrams, Alan Char, Mark Horton, and Joe Kalash.

"YACC: Yet Another Compiler-Compiler" acknowledgements: B.W. Kernighan, P.J. Plauger, S.I. Feldman, C. Imagna, M.E. Lesk, A. Snyder, C.B. Haley, D.M. Ritchie, M.D. Harris and Al Aho, for their ideas and assistance.

"Lex – A Lexical Analyzer Generator" acknowledgements: S.C. Johnson, A.V. Aho, and Eric Schmidt, for their help as originators of much of Lex, as well as debuggers of it.

The document "RATFOR – A Preprocessor for a Rational Fortran" is a revised and expanded version of the one published in Software – Practice and Experience, October 1975. The Ratfor described here is the one in use on UNIX and GCOS at AT&T Bell Laboratories. Acknowledgements: Dennis Ritchie, and Stuart Feldman, for their ideas and assistance.

"The M4 Macro Processor" acknowledgements: Rick Becker, John Chambers, Doug McIlroy, and Jim Weythman, for their help and support.

"BC – An Arbitrary Precision Desk-Calculator Language" acknowledgement: The compiler is written in YACC; its original version was written by S.C. Johnson.

"A Dial-Up Network of UNIX TM Systems" acknowledgements: G.L. Chesson, A.S. Cohen, J. Lions, and P.F. Long, for their suggestions and assistance.

The document "Berkeley Pascal User's Manual" is copyrighted © 1977, 1979, 1980, 1983 by W.N. Joy, S.L. Graham, C.B. Haley, M.K. McKusick, P.B. Kessler. The financial support of the first and second authors' work by the National Science Foundation under grants MCS74-07644-A04, MCS78-07291, and MCS80-05144, and the first author's work by an IBM Graduate Fellowship are gratefully acknowledged.

d The Fourth Berkeley Software Distribution is provided by the Regents of the University of California and the Other Contributors on an "as is" basis. Neither the Regents of the University of California nor the Other Contributors warrant that the functions contained in the Fourth Berkeley Software Distribution will meet the licensee's requirements or will operate in the combinations which may be selected for use by the licensee, or that the operation of the Fourth Berkeley Software Distribution will be uninterrupted or error free. Neither the Regents of the University of California nor the Other Contributors make any warranties, either express or implied, as to any matter whatsoever, including without limitation, the condition of the Fourth Berkeley Software Distribution, its merchantability or its fitness for any particular purpose.

e The licensee understands and agrees that the Regents of the University of California and the Other Contributors are under no obligations to provide either maintenance services, update services, notices of latent defects, or corrections of defects for Fourth Berkeley Software Distribution.

INTRODUCTION TO VOLUME 1

This volume gives descriptions of the publicly available features of the UNIX† and, therefore, the ULTRIX-11 system. It does not attempt to provide perspective or tutorial information upon the UNIX or the ULTRIX-11 operating system, its facilities, or its implementation. Various documents on those topics are contained in Volume 2, the ULTRIX-11 Software Technical Description, the ULTRIX-11 Installation Guide, and the ULTRIX-11 System Management Guide. In particular, for an overview see 'The UNIX Time-Sharing System' by Ritchie and Thompson; for a tutorial see 'UNIX for Beginners' by Kernighan.

Within the area it surveys, this volume attempts to be timely, complete and concise. Where the latter two objectives conflict, the obvious is often left unsaid in favor of brevity. It is intended that each program be described as it is, not as it should be. Inevitably, this means that various sections will soon be out of date.

The volume is divided into eight sections:

1. Commands
2. System calls
3. Subroutines
4. Special files
5. File formats and conventions
6. Games
7. Macro packages and language conventions
8. Maintenance

Commands are programs intended to be invoked directly by the user, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in directory /bin (for binary programs). Some programs also reside in /usr/bin, to save space in /bin. These directories are searched automatically by the command interpreter.

System calls are entries into the ULTRIX-11 supervisor. Every system call has one or more C language interfaces described in section 2. The underlying assembly language interface, coded with opcode sys, a synonym for trap, is given as well.

† UNIX is a trademark of AT&T Bell Laboratories.

An assortment of subroutines is available; they are described in section 3. The primary libraries in which they are kept are described in intro(3). The functions are described in terms of C, but most will work with Fortran as well.

The special files section 4 discusses the characteristics of each system 'file' that actually refers to an I/O device. The names in this section refer to the DEC device names for the hardware, instead of the names of the special files themselves.

The file formats and conventions section 5 documents the structure of particular kinds of files; for example, the form of the output of the loader and assembler is given. Excluded are files used by only one command, for example the assembler's intermediate files.

Games have been relegated to section 6 to keep them from contaminating the more staid information of section 1.

Section 7 is a miscellaneous collection of information necessary to writing in various specialized languages: character codes, macro packages for typesetting, etc.

The maintenance section 8 discusses procedures not intended for use by the ordinary user. These procedures often involve use of commands of section 1, where an attempt has been made to single out peculiarly maintenance-flavored commands by marking them 1M. Much of section 8 has been superseded by the ULTRIX-11 System Management Guide.

Each section consists of a number of independent entries of a page or so each. The name of the entry is in the upper corners of its pages, together with the section number, and sometimes a letter characteristic of a subcategory, e.g. graphics is 1G, and the math library is 3M. Entries within each section are alphabetized. The page numbers of each entry start at 1; it is infeasible to number consecutively the pages of a document like this that is republished in many variant forms.

All entries are based on a common format, not all of whose subsections will always appear.

The name subsection lists the exact names of the commands and subroutines covered under the entry and gives a very short description of their purpose.

The synopsis summarizes the use of the program being described. A few conventions are used, particularly in the Commands subsection:

Square brackets [ ] around an argument indicate that the argument is optional. When an argument is given as 'name', it always refers to a file name.

Ellipses '...' are used to show that the previous argument-prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus sign '-' is often taken to mean some sort of option-specifying argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with '-'.

The description subsection discusses in detail the subject at hand.

The files subsection gives the names of files which are built into the program.

A see also subsection gives pointers to related information.

A diagnostics subsection discusses the diagnostic indications which may be produced. Messages which are intended to be self-explanatory are not listed.

The bugs subsection gives known bugs and sometimes deficiencies. Occasionally also the suggested fix is described.

In section 2 an assembler subsection carries the assembly language system interface.

At the beginning of the volume is a table of contents, organized by section and alphabetically within each section. There is also a permuted index derived from the table of contents. Within each index entry, the title of the writeup to which it refers is followed by the appropriate section number in parentheses. This fact is important because there is considerable name duplication among the sections, arising principally from commands which exist only to exercise a particular system call.

HOW TO GET STARTED

This section sketches the basic information you need to get started on ULTRIX-11: how to log in and log out, how to communicate through your terminal, and how to run a program. See 'UNIX for Beginners' in Volume 2 for a more complete introduction to the system.

Logging in. You must call, or directly access, ULTRIX-11 from an appropriate terminal. Most hardcopy and video terminals, preferably with upper and lower case, are supported. You must also have a valid user name, which may be obtained, together with the telephone number, from the system administrators. The same telephone number serves terminals operating at all the standard speeds. After a data connection is established, the login procedure depends on what kind of terminal you are using.

300-baud terminals: These terminals generally have a speed switch which should be set at '300' (or '30' for 30 characters per second) and a half/full duplex switch which should be set at full-duplex. (This switch will often have to be changed since many other systems require half-duplex). When a connection is established, the system types 'login:'; you type your user name, followed by the 'return' key. If you have a password, the system asks for it and turns off the printer on the terminal so the password will not appear. After you have logged in, the 'return', 'new line', or 'linefeed' keys will give exactly the same results.

1200- and 150-baud terminals: If there is a half/full duplex switch, set it at full-duplex. When you have established a data connection, the system types out a few garbage characters (the 'login:' message at the wrong speed). Depress the 'break' (or 'interrupt') key; this is a speed-independent signal to ULTRIX-11/ that a different speed terminal is in use. The system then will type 'login:,' this time at another speed. Continue depressing the break key until 'login:' appears in clear, then respond with your user name. Terminate each line you type with the 'return' key.

Hard-wired terminals. Hard-wired terminals usually begin at the right speed, up to 9600 baud; otherwise the preceding instructions apply.

For all these terminals, it is important that you type your name in lower-case if possible; if you type upper-case letters, ULTRIX-11 will assume that your terminal cannot generate lower-case letters and will translate all subsequent upper-case letters to lower case.

The evidence that you have successfully logged in is that the Shell program will type a '$' to you. (The Shell is described below under 'How to run a program.')

For more information, consult stty(1), which tells how to adjust terminal behavior, getty(8), which discusses the login sequence in more detail, and tty(4), which discusses terminal I/O.

Logging out. There are three ways to log out:

You can simply hang up the phone.

You can log out by typing an end-of-file indication (EOT character, control-d) to the Shell. The Shell will terminate and the 'login: ' message will appear again.

You can also log in directly as another user by giving a login(1) command.

How to communicate through your terminal. When you type characters, a gnome deep in the system gathers your characters and saves them in a secret place. The characters will not be given to a program until you type a return (or newline), as described above in Logging in.

ULTRIX-11 terminal I/O is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the printed output will have the input characters interspersed. However, whatever you type will be saved up and interpreted in correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away all the saved characters.

The character 'control-u' in typed input kills all the preceding characters in the line, so typing mistakes can be repaired on a single line. Also, the character 'delete' erases the last character typed. Successive uses of 'delete' erase characters back to, but not beyond, the beginning of the line. The 'control-u' and 'delete' can be transmitted to a program by preceding them with '\'. (So, to erase '\', you need two 'delete's). These conventions can be changed by the stty(1) command.

Typing 'control-c' causes an interrupt signal, which is not passed to programs. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor, for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The quit signal is generated by typing the ASCII FS character. (FS appears many places on different terminals, most commonly as control-\.) It not only causes a running program to terminate but also generates a file with the core image of the terminated process. Quit is useful for debugging.

Besides adapting to the speed of the terminal, ULTRIX-11 tries to be intelligent about whether you have a terminal with the newline function or whether it must be simulated with carriage-return and line-feed. In the latter case, all input carriage returns are turned to newline characters (the standard line delimiter) and both a carriage return and a line feed are echoed to the terminal. If you get into the wrong mode, the stty(1) command will rescue you.

Tab characters are used freely in ULTRIX-11 source programs. If your terminal does not have the tab function, you can arrange to have them turned into spaces during output, and echoed as spaces during input. The system assumes that tabs are set every eight columns. Again, the stty(1) command will set or reset this mode. Also, the command tabs(1) will set the tab stops automatically on many terminals.

How to run a program; the Shell. When you have successfully logged in, a program called the Shell is listening to your terminal. The Shell reads typed-in lines, splits them up into a command name and arguments, and executes the command. A command is simply an executable program. The Shell looks first in your current directory (see below) for a program with the given name, and if none is there, then in a system directory. There is nothing special about system-provided commands except that they are kept in a directory where the Shell can find them.

The command name is always the first word on an input line; it and its arguments are separated from one another by spaces.

When a program terminates, the Shell will ordinarily regain control and type a '$' at you to indicate that it is ready for another command.

The Shell has many other capabilities, which are described in detail in section sh(1).

The current directory. ULTRIX-11 has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he also created a directory for you (ordinarily with the same name as your user name). When you log in, any file name you type is by default in this directory. Since you are the owner of this directory, you have full permission to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their owners. As a matter of observed fact, few ULTRIX-11 users protect their files from destruction, let alone perusal, by other users.

To change the current directory (but not the set of permissions you were endowed with at login) use cd(1).

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with '/', the name of the root directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a '/') until finally the file name is reached. For example, /usr/lem/filex refers to the file filex in the directory lem; lem is itself a subdirectory of usr; usr springs directly from the root directory.

If your current directory has subdirectories, the path names of files therein begin with the name of the subdirectory with no prefixed '/'.

A path name may be used anywhere a file name is required.

Important commands which modify the contents of files are cp(1), mv(1), and rm(1), which respectively copy, move (i.e. rename) and remove files. To find out the status of files or directories, use ls(1). See mkdir(1) for making directories and rmdir (in rm(1)) for destroying them.

For a fuller discussion of the file system, see 'The UNIX Time-Sharing System,' by Ken Thompson and Dennis Ritchie and Chapter 1 of the 'ULTRIX-11 System Management Guide'. It may also be useful to glance through section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Writing a program. To enter the text of a source program into a ULTRIX-11 file, use the editor ed(1). The three principal languages in ULTRIX-11 are provided by the C compiler cc(1), the Fortran compiler f77(1), and the assembler as(1). After the program text has been entered through the editor and written on a file, you can give the file to the appropriate language processor as an argument. The output of the language processor will be left on a file in the current directory named 'a.out'. (If the output is precious, use mv to move it to a less exposed name soon.) If you wrote in assembly language, you will probably need to load the program with library subroutines; see ld(1). The other two language processors call the loader automatically.

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the Shell in response to the '$' prompt.

Your programs can receive arguments from the command line just as system programs do, see exec(2).

Text processing. Almost all text is entered through the editor ed(1). The commands most often used to write text on a terminal are: cat, pr, roff and nroff, all in section 1.

The cat command simply dumps ASCII text on the terminal, with no processing at all. The pr command paginates the text, supplies headings, and has a facility for multi-column output. Nroff is an elaborate text formatting program. Used naked, it requires careful forethought, but for ordinary documents it has been tamed; see ms(7). Roff is a simpler text formatting program, and requires somewhat less forethought.

Troff prepares documents for a Graphics Systems phototypesetter; it is very similar to nroff, and often works from exactly the same source text. It was used to produce this manual.

Status inquiries. Various commands exist to provide you with useful information. Who(1) prints a list of users presently logged in. Date(1) prints the current time and date. Ls(1) will list the files in your directory or give summary information about particular files.

Surprises. Certain commands provide inter-user communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you.

To communicate with another user currently logged in, write(1) is used; mail(1) will leave a message whose presence will be announced to another user when he next logs in. The write-ups in the manual also suggest how to respond to the two commands if you are a target.

When you log in, a message-of-the-day may greet you before the first '$'.

CONVERTING FROM THE 6TH EDITION

There follows a catalogue of significant, mostly incompatible, changes that will affect old users converting to the 7th edition. No attempt is made to list all new facilities, or even all minor, but easily spotted changes, just the bare essentials without which it will be almost impossible to do anything.

Addressing files. Byte addresses in files are now long (32-bit) integers. Accordingly seek has been replaced by lseek(2). Every program that contains a seek must be modified. Stat and fstat(2) have been affected similarly, since file lengths are now 32- rather than 24-bit quantities.

Assembly language. System entry points are no longer built in symbols. Their values must be obtained from /usr/include/sys.s, see intro(2). All system calls modify r0. This means that sequences like

```
mov     file,r0
sys     lseek,0,0,2
sys     write,buf,n
```

will no longer work. (In fact, lseek now modifies r1 as well, so be doubly cautious.)

The sleep(2) entry point is gone; see the more general facility, alarm, plus pause.

Few library functions have assembly language entry points any more. You will have to simulate the C calling sequence.

Stty and gtty. These system calls have been extensively altered, see ioctl(2) and tty(4).

Archive files. The format of files produced by ar(1) has been altered. To convert to the new style, use arcv(1).

C language, lint. The official syntax for initialization requires an equal sign = before an initializer, and brackets { } around compound initial values; arrays and structures are now initialized honestly. Two-address operators, such as =+ and =-, are now written += and -= to avoid ambiguities, although the old style is still accepted. You will also certainly want to learn about

```
long integers
type definitions
casts (for type conversion)
unions (for more honest storage sharing)
#include <filename> (which searches in standard places)
```

The program lint(1) checks for obsolete syntax and does strong type checking of C programs, singly or in groups that are expected to be loaded together. It is indispensable for conversion work.

Fortran. The old fc is replaced by f77, a true compiler for Fortran 77, compatible with C. There are substantial changes in the language; see 'A Portable Fortran 77 Compiler' in Volume 2.

Stream editor. The program sed(1) is adapted to massive, repetitive editing jobs of the sort encountered in converting to the new system. It is well worth learning.

Standard I/O. The old fopen, getc, putc complex and the old -lp package are both dead, and even getchar has changed. All have been replaced by the clean, highly efficient, stdio(3) package. The first things to know are that getchar(3) returns the integer EOF (-1), which is not a possible byte value, on end of file, that 518-byte buffers are out, and that there is a defined FILE data type.

Make. The program make(1) handles the recompilation and loading of software in an orderly way from a 'makefile' recipe given for each piece of software. It remakes only as much as the modification dates of the input files show is necessary. The makefiles will guide you in building your new system.

Shell, chdir. F. L. Bauer once said Algol 68 is the Everest that must be climbed by every computer scientist because it is there. So it is with the shell for ULTRIX-11 users. Everything beyond simple command invocation from a terminal is different. Even chdir is now spelled cd. You will want to study sh(1) long and hard.

Debugging. Adb(1) is a far more capable replacement for the debugger db. The first-time user should be especially careful about distinguishing / and ? in adb commands, and watching to make sure that the x whose value he asked for is the real x, and not just some absolute location equal to the stack offset of some automatic x. You can always use the 'true' name, _x, to pin down a C external variable.

Dsw. This little-known, but indispensable facility has been taken over by rm -ri.

Boot procedures. Needless to say, these are all different. For further information, read Chapter 3 in the 'ULTRIX-11 System Management Guide'.

TABLE OF CONTENTS

SECTION 1. COMMANDS

SECTION 2. SYSTEM CALLS

SECTION 3. SUBROUTINES

SECTION 4. SPECIAL FILES

SECTION 5. FILE FORMATS AND CONVENTIONS

SECTION 6. GAMES

SECTION 7. MACRO PACKAGES AND LANGUAGE CONVENTIONS

SECTION 8. MAINTENANCE

PERMUTED INDEX

NAME

intro - introduction to commands

DESCRIPTION

This section describes publicly accessible commands in alphabetic order. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.

(1C) Commands for communication with other systems.

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

(1M) Commands used primarily for system maintenance.

The word 'local' at the foot of a page means that the command is not intended for general distribution.

SEE ALSO

DIAGNOSTICS

Section (6) for computer games.

How to get started, in the Introduction.

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.

NAME

adb - debugger

SYNOPSIS

adb [-w] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of adb cannot be used although the file can still be examined. The default for objfil is a.out. Corfil is assumed to be a core image file produced after executing objfil; the default for corfil is core.

Requests to adb are read from the standard input and responses are to the standard output. If the -w flag is present then both objfil and corfil are created if necessary and opened for reading and writing so that files can be modified using adb. Adb ignores QUIT; INTERRUPT causes return to the next adb command.

In general requests to adb are of the form

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

If address is present then dot is set to address. Initially dot is set to 0. For most commands count specifies how many times the command will be executed. The default count is 1. Address and count are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see ADDRESSES.

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.

integer
An octal number if integer begins with a 0; a hexadecimal number if preceded by #; otherwise a decimal number.

integer.fraction
A 32 bit floating point number.

'cccc' The ASCII value of up to 4 characters. \ may be used to escape a '.

< name The value of name, which is either a variable name or a register name. Adb maintains a number of variables (see VARIABLES) named by single letters or digits. If name is a register name then the value of the register is obtained from the system header in corfil. The register names are r0 ... r5 sp pc ps.

symbol A symbol is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the symbol is taken from the symbol table in objfil. An initial _ or ~ will be prepended to symbol if needed.

_ symbol
In C, the 'true name' of an external symbol begins with _. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

routine.name
The address of the variable name in the specified C routine. Both routine and name are symbols. If name is omitted the value is the address of the most recently activated C stack frame corresponding to routine.

(exp) The value of the expression exp.

Monadic operators

*exp The contents of the location addressed by exp in corfil.

@exp The contents of the location addressed by exp in objfil.

-exp Integer negation.

~exp Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1+e2 Integer addition.

e1-e2 Integer subtraction.

e1*e2 Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

e1#e2 E1 rounded up to the next multiple of e2.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '*'; see ADDRESSES for further details.)

?f Locations starting at address in objfil are printed according to the format f.

/f Locations starting at address in corfil are printed according to the format f.

=f The value of address itself is printed in the styles indicated by the format f. (For i format '?' is printed for the parts of the instruction that reference subsequent words.)

A format consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format dot is incremented temporarily by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

o 2 Print 2 bytes in octal. All octal numbers output by adb are preceded by 0.
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.
u 2 Print as an unsigned decimal number.

ADB(1)

U 4 Print long unsigned decimal.
f 4 Print the 32 bit value as a floating point number.
F 8 Print double floating point.
b 1 Print the addressed byte in octal.
c 1 Print the addressed character.
C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
s n Print the addressed characters until a zero character is reached.
S n Print a string using the @ escape convention. n is the length of the string including its zero terminator.
Y 4 Print 4 bytes in date format (see ctime(3)).
i n Print as PDP11 instructions. n is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a 0 Print the value of dot in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/ local or global data symbol
? local or global text symbol
= local or global absolute symbol

p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as a.
t 0 When preceded by an integer tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop.
r 0 Print a space.
n 0 Print a newline.
"..." 0
Print the enclosed string.
^ Dot is decremented by the current increment. Nothing is printed.
+ Dot is incremented by 1. Nothing is printed.
- Dot is decremented by 1. Nothing is printed.

newline
If the previous command temporarily incremented dot, make the increment permanent. Repeat the previous command with a count of 1.

[?/]l value mask
Words starting at dot are masked with mask and compared with value until a match is found. If L is used then the match is for 4 bytes at a time instead of 2. If no

match is found then dot is unchanged; otherwise dot is set to the matched location. If mask is omitted then -1 is used.

[?/]w value ...
Write the 2-byte value into the addressed location. If the command is W, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m b1 e1 f1[?/]
New values for (b1, e1, f1) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the '?' or '/' is followed by '*' then the second segment (b2,e2,f2) of the mapping is changed. If the list is terminated by '?' or '/' then the file (objfil or corfil respectively) is used for subsequent requests. (So that, for example, '/m?' will cause '/' to refer to objfil.)

>name
Dot is assigned to the variable or register named.

&modifier
The & modifier allows adb to deal with locations in overlay text segments, i.e., type 0430 and 0431 files. Locations is overlay text segments must be accessed by a symbol name or a symbol name plus an offset. For example, to display the contents of location (lpintr) as an instruction would be (lpintr/*i) for a non-overlaid file. The command (lpintr/*&i) would be used of the symbol is located in an overlay text segment.

! A shell is called to read the rest of the line following '!'.

$modifier
Miscellaneous commands. The available modifiers are:

- <f Read commands from the file f and return.
- >f Send output to the file f, which is created if it does not exist.
- r Print the general registers and the instruction addressed by pc. Dot is set to pc.
- f Print the floating registers in single or double length. If the floating point status of ps is set to double (0200 bit) then double length is used anyway.
- b Print all breakpoints and their associated counts and commands.
- a ALGOL 68 stack backtrace. If address is given then it is taken to be the address of the current frame (instead of r4). If count is given then

ADB(1)

only the first count frames are printed.

c C stack backtrace. If address is given then it is taken as the address of the current frame (instead of r5). If C is used then the names and (16 bit) values of all automatic and static variables are printed for each active function. If count is given then only the first count frames are printed.

e The names and values of external variables are printed.

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.

d Reset integer input as described in EXPRESSIONS.

q Exit from adb.

v Print all non zero variables in octal.

m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

bc Set breakpoint at address. The breakpoint is executed count-1 times before causing a stop. Each time the breakpoint is encountered the command c is executed. If this command sets dot to zero then the breakpoint causes a stop.

d Delete breakpoint at address.

r Run objfil as a subprocess. If address is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. count specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.

cs The subprocess is continued with signal s c s, see signal(2). If address is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for r.

ss As for c except that the subprocess is single stepped count times. If there is no current subprocess then objfil is run as a subprocess as for

r. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by adb but are not used subsequently. Numbered variables are reserved for communication as follows.

0 The last value printed.
1 The last offset part of an instruction source.
2 The previous value of variable 1.

On entry the following are set from the system header in the corfil. If corfil does not appear to be a core file then these values are set from objfil.

b The base address of the data segment.
d The data segment size.
e The entry point.
m The 'magic' number (0405, 0407, 0410 or 0411).
s The stack segment size.
t The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (b1, e1, f1) and (b2, e2, f2) and the file address corresponding to a written address is calculated as follows.

b1< address<e1 => file address=address+f1-b1, otherwise,

b2< address<e2 => file address=address+f2-b2,

otherwise, the requested address is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a ? or / is followed by an * then only the second triple is used.

The initial setting of both mappings is suitable for normal a.out and core files. If either file is not of the kind expected then, for that file, b1 is set to 0, e1 is set to the maximum file size and f1 is set to 0; in this way the whole file can be examined with no address translation.

So that adb may be used on large files all appropriate values are kept as signed 32 bit integers.

ADB(1)

FILES

/dev/mem
/dev/swap
a.out
core

SEE ALSO

ptrace(2), a.out(5), core(5), Setting up Unix

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.

RESTRICTIONS

A breakpoint set at the entry point is not effective on initial entry to the program.
When single stepping, system calls do not count as an executed instruction.
Local variables whose names are the same as an external variable may foul up the accessing of the external.

NAME

admin - create and administer SCCS files

SYNOPSIS

admin [-n] [-i[name]] [-rrel] [-t[name]]
[-fflag[flag-val]] [-dflag[flag-val]]
[-alogin] [-elogin] [-m[mrlist]] [-y[comment]] [-h] [-z]
files

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to admin, which may appear in any order, consist of keyletter arguments, which begin with -, and named files (note that SCCS file names must begin with the characters s.). If a named file doesn't exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, 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 (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 keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

-n This keyletter indicates that a new SCCS file is to be created.

-i[name] The name of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see -r keyletter for delta numbering scheme). If the i keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an admin command on which the i keyletter is supplied. Using a single admin to

ADMIN(1)

create two or more SCCS files require that they be created empty (no -i keyletter). Note that the -i keyletter implies the -n keyletter.

-r*rel* The *rel*ease into which the initial delta is inserted. This keyletter may be used only if the -i keyletter is also used. If the -r keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

-t[*name*] The *name* of a file from which descriptive text for the SCCS file is to be taken. If the -t keyletter is used and *admin* is creating a new SCCS file (the -n and/or -i keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a -t keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a -t keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

-f*flag* This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several f keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:

b Allows use of the -b keyletter on a *get*(1) command to create branch deltas.

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

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

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

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

j Allows concurrent get(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

llist A list of releases to which deltas can no longer be made (get -e against one of these 'locked' releases fails). 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 Causes delta(1) to create a 'null' delta in each of those releases (if any) being skipped when a delta is made in a new release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as 'anchor points' so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file preventing branch deltas from being created from them in the future.

qtext User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by get(1).

mmod Module name of the SCCS file substituted for all occurrences of the %M% keyword in 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.

ADMIN(1)

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

v[pgm] Causes delta(1) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value 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 keyletter must also be used even if its value is null).

-dflag Causes removal (deletion) of the specified flag from an SCCS file. The -d keyletter may be specified only when processing existing SCCS files. Several -d keyletters may be supplied on a single admin command. See the -f keyletter for allowable flag names.

llist A list of releases to be 'unlocked'. See the -f keyletter for a description of the l flag and the syntax of a list.

-alogin A login name, or numerical UNIX group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all login names common to that group ID. Several a keyletters may be used on a single admin command line. As many logins, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.

-elogin A login name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all login names common to that group ID. Several e keyletters may be used on a single admin command line.

-y[comment] The comment text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of delta(1). Omission of the -y keyletter results in a default comment line being

inserted in the form:
date and time created YY/MM/DD HH:MM:SS
by login
The -y keyletter is valid only if the -i and/or -n keyletters are specified (i.e., a new SCCS file is being created).

-m[mrlist] The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to delta(1). The v flag must be set and the MR numbers are validated if the v flag has a value (the name of an MR number validation program). Diagnostics will occur if the v flag is not set or MR validation fails.

-h Causes admin to check the structure of the SCCS file (see sccsfile(5)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see -h, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

FILES

The last component of all SCCS file names must be of the form s.file-name. New SCCS files are given mode 444 (see chmod(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by admin is to a temporary x-file, called x.file-name, (see get(1)), created with mode 444 if the admin command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of admin, the SCCS file is removed (if it exists), and the x-file is

renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by 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 allowing use of ed(1). Care must be taken! 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 the SCCS file is valid.

Admin also makes use of a transient lock file (called z.file-name), which is used to prevent simultaneous updates to the SCCS file by different users. See get(1) for further information.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use help(1) for explanations.

NAME

ar - archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the loader. It can be used, though, for any similar purpose.

Key is one character from the set drqtpmx, optionally concatenated with one or more of vuaibcl. Afile is the archive file. The names are constituent files in the archive file. The meanings of the key characters are:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character u is used with r, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set abi is used, then the posname argument must be present and specifies that new files are to be placed after (a) or before (b or i) posname. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the posname argument must be present and, as in r, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does x alter the archive file.

v Verbose. Under the verbose option, ar gives a file-by-file description of the making of a new archive file

AR(1)

from the old archive and the constituent files. When used with t, it gives a long listing of all information about the files. When used with p, it precedes each file with a name.

c Create. Normally ar will create afile when it needs to. The create option suppresses the normal message that is produced when afile is created.

l Local. Normally ar places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

FILES

/tmp/v* temporaries

SEE ALSO

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

RESTRICTIONS

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

NAME

arcv - convert archives to new format

SYNOPSIS

arcv file ...

DESCRIPTION

Arcv converts archive files (see ar(1), ar(5)) from 6th edition to 7th edition format. The conversion is done in place, and the command refuses to alter a file not in old archive format.

Old archives are marked with a magic number of 0177555 at the start; new archives have 0177545.

FILES

/tmp/v*, temporary copy

SEE ALSO

ar(1), ar(5)

NAME

as - assembler

SYNOPSIS

as [-] [-V] [-o objfile] file ...

DESCRIPTION

As assembles the concatenation of the named files. If the optional first argument - is used, all undefined symbols in the assembly are treated as global. If the optional -V argument is used, references to global text symbols are not resolved, but left for the loader to resolve. This should be done for files that are to be placed in overlays. It does not hurt to use the -V flag as long as the output is run through the loader, but failure to use it for modules that go into overlays can cause improper text references.

The output of the assembly is left on the file objfile; if that is omitted, a.out is used. It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

FILES

/lib/as2	pass 2 of the assembler
/tmp/atm[1-3]?	temporary
a.out	object

SEE ALSO

ld(1), nm(1), adb(1), a.out(5)
UNIX Assembler Manual by D. M. Ritchie

DIAGNOSTICS

When an input file cannot be read, its name followed by a question mark is typed and assembly ceases. When syntactic or semantic errors occur, a single-character diagnostic is typed out together with the line number and the file name in which it occurred. Errors in pass 1 cause cancellation of pass 2. The possible errors are:

)	Parentheses error
]	Parentheses error
<	String not terminated properly
*	Indirection used illegally
.	Illegal assignment to '.'
a	Error in address
b	Branch instruction is odd or too remote
e	Error in expression
f	Error in local ('f' or 'b') type symbol
g	Garbage (unknown) character
i	End of file inside an if
m	Multiply defined symbol as label
o	Word quantity assembled at odd address

p '.' different in pass 1 and 2
r Relocation error
u Undefined symbol
x Syntax error

RESTRICTIONS
Syntax errors can cause incorrect line numbers in following diagnostics.

NAME

at - execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

At squirrels away a copy of the named file (standard input default) to be used as input to sh(1) at a specified later time. A cd(1) command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the user and group ID of the creator of the copy file.

The time is 1 to 4 digits, with an optional following 'A', 'P', 'N' or 'M' for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional day is either (1) a month name followed by a day number, or (2) a day of the week; if the word 'week' follows invocation is moved seven days further off. Names of months and days may be recognizably truncated. Examples of legitimate commands are

```
at 8am jan 24
at 1530 fr week
```

At programs are executed by periodic execution of the command /usr/lib/atrun from cron(8). The granularity of at depends upon how often atrun is executed.

Standard output or error output is lost unless redirected.

FILES

/usr/spool/at/yy.ddd.hhhh.uu
activity to be performed at hour hhhh of year day ddd of year yy. uu is a unique number.
/usr/spool/at/lasttimedone contains hhhh for last hour of activity.
/usr/spool/at/past directory of activities now in progress
/usr/lib/atrun program that executes activities that are due
pwd(1)

SEE ALSO
calendar(1), cron(8)

DIAGNOSTICS
Complains about various syntax errors and times out of range.

RESTRICTIONS
Due to the granularity of the execution of /usr/lib/atrun, there may be bugs in scheduling things almost exactly 24 hours into the future.

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [-F*c*] [prog] [file] ...

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as -f *file*.

Files are read in order; if there are no files, the standard input is read. The file name '-' means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS, *vide infra*.) The fields are denoted $1, $2, ... ; $0 refers to the entire line.

A pattern-action statement has the form

```
pattern { action }
```

A missing { action } means print the line; a missing pattern always matches.

An action is a sequence of statements. 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
```

Statements are terminated by semicolons, newlines 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. Variables 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; this allows for a form of associative memory. String constants are quoted "...".

The print statement prints its arguments on the standard output (or on a file if >file 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 (see printf(3)).

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. The last truncates its argument to an integer. substr(s, m, n) returns the n-character substring of s that begins at position m. The function sprintf(fmt, expr, expr, ...) formats the expressions according to the printf(3) format given by fmt and returns the resulting string.

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. 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 ~ (for contains) or !~ (for does not contain). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

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

```
BEGIN { FS = "c" }
```

or by using the -Fc 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

lex(1), sed(1)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, Awk - a pattern scanning and processing language

RESTRICTIONS

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 "" to it.

NAME

bas - basic

SYNOPSIS

bas [file]

DESCRIPTION

Bas is a dialect of Basic. If a file argument is provided, the file is used for input before the terminal is read. Bas accepts lines of the form:

```
statement
integer statement
```

Integer numbered statements (known as internal statements) are stored for later execution. They are stored in sorted ascending order. Non-numbered statements are immediately executed. The result of an immediate expression statement (that does not have '=' as its highest operator) is printed. Interrupts suspend computation.

Statements have the following syntax:

expression
The expression is executed for its side effects (assignment or function call) or for printing as described above.

comment ...
This statement is ignored. It is used to interject commentary in a program.

done
Return to system level.

dump
The name and current value of every variable is printed.

edit
The UNIX editor, ed, is invoked with the file argument. After the editor exits, this file is recompiled.

for name = expression expression statement
for name = expression expression
...
next
The for statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of

BAS(1)

the second expression.

goto expression
The expression is evaluated, truncated to an integer and execution goes to the corresponding integer numbered statement. If executed from immediate mode, the internal statements are compiled first.

if expression statement
if expression
...
[else
...]
fi
The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. In the second form, an optional else allows for a group of statements to be executed when the first group is not.

list [expression [expression]]
is used to print out the stored internal statements. If no arguments are given, all internal statements are printed. If one argument is given, only that internal statement is listed. If two arguments are given, all internal statements inclusively between the arguments are printed.

print list
The list of expressions and strings are concatenated and printed. (A string is delimited by " characters.)

prompt list
Prompt is the same as print except that no newline character is printed.

return [expression]
The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run
The internal statements are compiled. The symbol table is re-initialized. The random number generator is reset. Control is passed to the lowest numbered internal statement.

save [expression [expression]]
Save is like list except that the output is written on the file argument. If no argument is given on the command, b.out is used.

Expressions have the following syntax:

name
A name is used to specify a variable. Names are composed of a letter followed by letters and digits. The first four characters of a name are significant.

number
A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an e followed by a possibly signed exponent.

(expression)
Parentheses are used to alter normal order of evaluation.

_ expression
The result is the negation of the expression.

expression operator expression
Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. A complete list of operators is given below.

expression ([expression [, expression] ...])
Functions of an arbitrary number of arguments can be called by an expression followed by the arguments in parentheses separated by commas. The expression evaluates to the line number of the entry of the function in the internally stored statements. This causes the internal statements to be compiled. If the expression evaluates negative, a builtin function is called. The list of builtin functions appears below.

name [expression [, expression] ...]
Each expression is truncated to an integer and used as a specifier for the name. The result is syntactically identical to a name. a[1,2] is the same as a[1][2]. The truncated expressions are restricted to values between 0 and 32767.

The following is the list of operators:

= = is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left,

& | & (logical and) has result zero if either of its arguments are zero. It has result one if both its arguments are non-zero. | (logical or) has result zero if

BAS(1)

both of its arguments are zero. It has result one if either of its arguments are non-zero.

< <= > >= == <>
The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, <> not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: a>b>c is the same as a>b&b>c.

+ - Add and subtract.

* / Multiply and divide.

^ Exponentiation.

The following is a list of builtin functions:

arg(i) is the value of the i -th actual parameter on the current level of function call.

exp(x) is the exponential function of x.

log(x) is the natural logarithm of x.

sqr(x) is the square root of x.

sin(x) is the sine of x (radians).

cos(x) is the cosine of x (radians).

atn(x) is the arctangent of x. Its value is between -J/2 and J/2.

rnd( ) is a uniformly distributed random number between zero and one.

expr( )
is the only form of program input. A line is read from the input and evaluated as an expression. The resultant value is returned.

abs(x) is the absolute value of x.

int(x) returns x truncated (towards 0) to an integer.

FILES

/tmp/btm? temporary
b.out save file
/bin/ed for edit

DIAGNOSTICS

Syntax errors cause the incorrect line to be typed with an underscore where the parse failed. All other diagnostics are self explanatory.

RESTRICTIONS

Has been known to give core images.
Catches interrupts even when they are turned off.

NAME

basename - strip filename affixes

SYNOPSIS

basename string [suffix]

DESCRIPTION

Basename deletes any prefix ending in '/' and the suffix, if present in string, from string, and prints the result on the standard output. It is normally used inside substitution marks ` ` in shell procedures.

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

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

SEE ALSO

sh(1)

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

```
bc [ -c ] [ -l ] [ file ... ]
```

DESCRIPTION

Bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The -l argument stands for the name of an arbitrary precision math library. The syntax for bc programs is as follows; L means letter a-z, E means expression, S means statement.

Comments

are enclosed in /* and */.

Names

```
simple variables: L
array elements: L [ E ]
The words 'ibase', 'obase', and 'scale'
```

Other operands

```
arbitrarily long numbers with optional sign and
decimal point.
( E )
sqrt ( E )
length ( E )    number of significant decimal digits
scale ( E )     number of digits right of decimal point
L ( E , ... , E )
```

Operators

```
+  -  *  /  %  ^ (% is remainder; ^ is power)
++   --          (prefix and postfix; apply to names)
==  <=  >=  !=  <  >
=  =+  =-  =*  =/  =%  =^
```

Statements

```
E
{ S ; ... ; S }
if ( E ) S
while ( E ) S
for ( E ; E ; E ) S
null statement
break
quit
```

Function definitions

```
define L ( L ,..., L ) {
     auto L, ... , L
     S; ... S
```

```
        return ( E )
    }
```

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.

For example

```
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
     }
}
```

defines a function to compute an approximate value of the exponential function and

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

prints approximate values of the exponential function of the first ten integers.

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.

FILES

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

SEE ALSO

dc(1)
L. L. Cherry and R. Morris, BC - An arbitrary precision desk-calculator language

RESTRICTIONS

No &&, ||, or ! operators.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

BDIFF(1)

NAME

bdiff - big diff

SYNOPSIS

bdiff file1 file2 [n] [-s]

DESCRIPTION

Bdiff is used in a manner analogous to diff(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for diff. Bdiff ignores lines common to the beginning of both files, splits the remainder of each file into n-line segments, and invokes diff upon corresponding segments. The value of n is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for n. This is useful in those cases in which 3500-line segments are too large for diff, causing it to fail. If file1 (file2) is -, the standard input is read. The optional -s (silent) argument specifies that no diagnostics are to be printed by bdiff (note, however, that this does not suppress possible exclamations by diff. If both optional arguments are specified, they must appear in the order indicated above.

The output of bdiff is exactly that of diff, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, bdiff does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use help(1) for explanations.

NAME

bufstat - print I/O buffer usage

SYNOPSIS

bufstat [ifn] [interval] [corefile] [namelist]

DESCRIPTION

The bufstat printout shows the current usage of the system's I/O buffer cache. The optional argument 'i' specifies that the printout should be repeated every interval seconds. The 'f' option causes corefile to be used as memory instead of /dev/mem. The 'n' option declares that the system namelist should be obtained from namelist instead of /unix. Single or multiple options may be specified. The arguments interval, corefile, and namelist must occur in the same order as the options [ifn].

Information printed by bufstat includes; Physical memory address of the buffer, number of I/O operations on the buffer since system startup, the byte count and logical block number for the last I/O operation on the buffer, the error code, the 'name' of the device that owns the buffer, and the buffer header flags. Where 'name' is the special file name for the device, e.g., hk00, rl0, etc., or the device's major/minor device number, if the special file cannot be located.

FILES

/unix - default system namelist
/dev/mem - default system memory

SEE ALSO

Refer to the ULTRIX-11 System Management Guide, Section 9.5.2 for an example of a bufstat printout.

CAL(1)

NAME

cal - print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. Year can be between 1 and 9999. The month is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

RESTRICTIONS

The year is always considered to start in January even though this is historically naive.
Beware that 'cal 78' refers to the early Christian era, not the 20th century.

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

Calendar consults the file 'calendar' in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as 'Dec. 7,' 'december 7,' '12/7,' etc., are recognized, but not '7 December' or '7/12'. On weekends 'tomorrow' extends through Monday.

When an 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).

FILES

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

SEE ALSO

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

RESTRICTIONS

Your calendar must be public information for you to get reminder service.
Calendar's extended idea of 'tomorrow' doesn't account for holidays.

CAT(1)

NAME

cat - catenate and print

SYNOPSIS

cat [-u] file ...

DESCRIPTION

Cat reads each file in sequence and writes it on the standard output. Thus

```
cat file
```

prints the file and

```
cat file1 file2 >file3
```

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

If no file is given, or if the argument '-' is encountered, cat reads from the standard input. Output is buffered in 512-byte blocks unless the standard output is a terminal or the -u option is present.

SEE ALSO

pr(1), cp(1)

RESTRICTIONS

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

NAME

cb - C program beautifier

SYNOPSIS

cb

DESCRIPTION

Cb places a copy of the C program from the standard input on the standard output with spacing and indentation that displays the structure of the program.

RESTRICTIONS

CC(1)

NAME

cc, pcc - C compiler

SYNOPSIS

cc [option] ... file ...

pcc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler. It accepts several types of arguments:

Arguments whose names end with '.c' are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with '.o' substituted for '.c'. The '.o' file is normally deleted, however, if a single C program is compiled and loaded all at one go.

In the same way, arguments whose names end with '.s' are taken to be assembly source programs and are assembled, producing a '.o' file.

The following options are interpreted by cc. See ld(1) for load-time options.

-c Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.

-p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls monitor(3) at the start and arranges to write out a mon.out file at normal termination of execution of the object program. An execution profile can then be generated by use of prof(1).

-f In systems without hardware floating-point, use a version of the C compiler which handles floating-point constants and loads the object program with the floating-point interpreter. Do not use if the hardware is present.

-O Invoke an object-code optimizer.

-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed '.s'.

-P Run only the macro preprocessor and place the result

for each '.c' file in a corresponding '.i' file and has no '#' lines in it.

-E Run only the macro preprocessor and send the result to the standard output. The output is intended for compiler debugging; it is unacceptable as input to cc.

-V Create code suitable for use in overlaid programs. This is now a default flag.

-V7 Turn of the -V flag. This is provided only for backwards compatibility with older versions of cc, and in general should not be used.

-N Place switch tables into text space. This is useful for 430 type programs, since the switch tables then go into the overlay with the module, saving valuable data space. This option must NOT be used for separate instruction and data space programs!

-o output
Name the final output file output. If this option is used the file 'a.out' will be left undisturbed.

-Dname=def
-Dname Define the name to the preprocessor, as if by '#define'. If no definition is given, the name is defined as 1.

-Uname Remove any initial definition of name.

-Idir '#include' files whose names do not begin with '/' are always sought first in the directory of the file argument, then in directories named in -I options, then in directories on a standard list.

-Bstring
Find substitute compiler passes in the files named string with the suffixes cpp, c0, c1 and c2. If string is empty, use a standard backup version.

-t[p012]
Find only the designated compiler passes in the files whose names are constructed by a -B option. In the absence of a -B option, the string is taken to be '/usr/src/cmd/c/'.

Other arguments are taken to be either loader option arguments, or C-compatible object programs, typically produced by an earlier cc run, or perhaps libraries of C-compatible routines. These programs, together with the results of any

CC(1)

compilations specified, are loaded (in the order given) to produce an executable program with name a.out.

The major purpose of the 'portable C compiler', pcc, is to serve as a model on which to base other compilers. Pcc does not support options -f, -E, -B, and -t. It provides, in addition to the language of cc, unsigned char type data and initialized bit fields.

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporaries for cc
/lib/cpp	preprocessor
/lib/c[01]	compiler for cc
/usr/c/oc[012]	backup compiler for cc
/usr/c/ocpp	backup preprocessor
/lib/fc[01]	floating-point compiler
/lib/c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/fcrt0.o	startoff for floating-point interpretation
/lib/libc.a	standard library, see intro(3)
/usr/include	standard directory for '#include' files
/tmp/pc*	temporaries for pcc
/usr/lib/ccom	compiler for pcc

SEE ALSO

B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, 1978
D. M. Ritchie, C Reference Manual
monitor(3), prof(1), adb(1), ld(1)

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader. Of these, the most mystifying are from the assembler, as(1), in particular 'm', which means a multiply-defined external symbol (function or data).

RESTRICTIONS

Pcc is little tried on the PDP11; specialized code generated for that machine has not been well shaken down. The -O optimizer was designed to work with cc; its use with pcc is suspect.

NAME
cd - change working directory

SYNOPSIS
cd directory

DESCRIPTION
Directory becomes the new working directory. The process must have execute (search) permission in directory.

Because a new process is created to execute each command, cd would be ineffective if it were written as a normal command. It is therefore recognized and executed by the Shell.

SEE ALSO
sh(1), pwd(1), chdir(2)

CDA(1M)

NAME

cda, ccd - crash dump analysis

SYNOPSIS

```
cda -options [namelist] [corefile]
ccd
```

DESCRIPTION

The ccd command copies a crash dump from a magtape, RX50 diskette, or the swap area to a file, for analysis by the cda program.

Refer to the ULTRIX-11 System Management Guide, Section 9.5 for a description of the cda program and Section 9.4 for a description of the ccd program.

SEE ALSO

ULTRIX-11 System Management Guide, Chapter 9

NAME

cdc - change the delta commentary of an SCCS delta

SYNOPSIS

cdc -rSID [-m[mrlist]] [-y[comment]] files

DESCRIPTION

Cdc changes the delta commentary, for the SID specified by the -r keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the delta(1) command (-m and -y keyletters).

If a directory is named, cdc 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.

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

All the described keyletter arguments apply independently to each named file:

-rSID — Used to specify the SCCS IDentification (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)) then a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the -r keyletter may be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of delta(1). In order to delete an MR, precede the MR number with the character ! (see EXAMPLES). 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 -y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value (see admin(1)), 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 the comment(s) already existing for the delta specified by the -r keyletter. 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 new-line character terminates the comment text.

The exact permissions necessary to modify the SCCS file are documented in the Source Code Control System User's Guide. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

```
cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrou-
ble 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.

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

does the same thing.

WARNINGS

If SCCS file names are supplied to the cdc command via the standard input (- on the command line), then the -m and -y keyletters must also be used.

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use help(1) for explanations.

NAME

chfn - change finger entry

SYNOPSIS

chfn [loginname]

DESCRIPTION

Chfn is used to change information about users. This information is used by the finger program, among others. It consists of the user's "real life" name, office room number, office phone number, and home phone number. Chfn prompts the user for each field. Included in the prompt is a default value, which is enclosed between brackets. The default value is accepted simply by typing <return>. To enter a blank field, type the word 'none'. Below is a sample run:

```
Name [Biff Studsworth II]:
Office location []: 521E
Office Phone (Ex: 1632) []: 1863
Home Phone (Ex: 987532) [5771546]: none
```

Chfn allows phone numbers to be entered with or without hyphens.

It is a good idea to run finger after running chfn to make sure everything is the way you want it.

The optional argument loginname is used to change another person's finger information. This can only be done by the super-user.

FILES

/etc/passwd, /etc/ptmp

SEE ALSO

finger(1), passwd(5)

RESTRICTIONS

For historical reasons, the user's name, etc. are stored in the passwd file. This is a bad place to store the information.

Because two users may try to write the passwd file at once, a synchronization method was developed. On rare occasions, a message that the password file is "busy" will be printed. In this case, chfn sleeps for a while and then tries to write to the passwd file again.

NAME

chghist change the history entry of an SCCS delta

SYNOPSIS

chghist -rSID name ...

DESCRIPTION

Chghist changes the history information, for the delta specified by the SID, of each named SCCS file.

If a directory is named, chghist 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 exact permissions necessary to change the history entry of a delta are documented in the SCCS/PWB User's Manual. Simply stated, they are either (1) if you made a delta, you can change its history entry; or (2) if you own the file and directory you can change a history entry.

The new history is read from the standard input. If the standard input is a terminal, the program will prompt with 'MRs? ' (only if the file has a v flag, see admin (I)) and with 'comments? '. If the standard input is not a terminal, no prompt(s) is (are) printed. A newline preceded by a '\' is read as a blank, and may be used to make the entering of the history more convenient. The first newline not preceded by a '\' terminates the response for the corresponding prompt.

When the history entry of a delta table record (see prt (I)) is changed, all old MR entries (if any) are converted to comments, and both these and the original comments are preceded by a comment line that indicates who made the change and when it was made. The new information is entered preceding the old. No other changes are made to the delta table entry.

FILES

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

SEE ALSO

admin(1), get(1), delta(1), prt(1), help(1), sccsfile(5)
SCCS/PWB User's Manual by L. E. Bonanni and A. L. Glasser.

DIAGNOSTICS

Use help (1) for explanations.

NAME

chmod - change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

The mode of each named file is changed according to mode, which may be absolute or symbolic. An absolute mode is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see chmod(2)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic mode has the form:

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

The who part is a combination of the letters u (for user's permissions), g (group) and o (other). The letter a stands for ugo. If who is omitted, the default is a but the setting of the file creation mask (see umask(2)) is taken into account.

Op can be + to add permission to the file's mode, - to take away permission and = to assign permission absolutely (all other bits will be reset).

Permission is any combination of the letters r (read), w (write), x (execute), s (set owner or group id) and t (save text - sticky). Letters u, g or o indicate that permission is to be taken from the current mode. Omitting permission is only useful with = to take away all permissions.

The first example denies write permission to others, the second makes a file executable:

```
chmod o-w file
chmod +x file
```

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter s is only useful with u or g.

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

SEE ALSO
ls(1), chmod(2), chown (1), stat(2), umask(2)

CHOG(1)

NAME

chog - change owner and group

SYNOPSIS

chog owner files...

DESCRIPTION

Chog changes the user-ID and the group-ID of files to the user-ID and default group-ID of owner. The owner may be either a login name or a decimal UID, but it must be found in the password file.

Only the super-user is allowed to change owner and group ID's.

FILES

/etc/passwd
/etc/group

SEE ALSO

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

NAME

chown, chgrp - change owner or group

SYNOPSIS

```
chown owner file ...

chgrp group file ...
```

DESCRIPTION

Chown changes the owner of the files to owner. The owner may be either a decimal UID or a login name found in the password file.

Chgrp changes the group-ID of the files to group. The group may be either a decimal GID or a group name found in the group-ID file.

Only the super-user can change owner or group, in order to simplify as yet unimplemented accounting procedures.

FILES

```
/etc/passwd
/etc/group
```

SEE ALSO

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

CHROOT(1)

NAME

chroot - change root directory

SYNOPSIS

chroot newdir

DESCRIPTION

Chroot changes the root directory to newdir and then execs a new shell. The new shell is determined from the environment variable SHELL, if it is not set then /bin/sh is used. The shell is executed relative to the new directory. The prompt is changed to (subroot)-> to remind the user of his new environment.

Only the super-user can change the root directory.

FILES

/bin/sh

SEE ALSO

chroot(2)

NAME

chsh - change default login shell

SYNOPSIS

chsh name [shell]

DESCRIPTION

Chsh is a command similar to passwd(1) except that it is used to change the login shell field of the password file rather than the password entry. If no shell is specified then the shell reverts to the default login shell /bin/sh. Currently only /bin/csh or /bin/oldcsh can be specified as the shell unless you are the super-user.

An example use of this command would be

```
chsh bill /bin/csh
```

This example changes the default shell for the user bill to be /bin/csh.

SEE ALSO

csh(1), passwd(1), passwd(5)

CLRI(1M)

NAME

clri - clear i-node

SYNOPSIS

clri filesystem i-number ...

DESCRIPTION

Clri writes zeros on the i-nodes with the decimal i-numbers on the filesystem. After clri, any blocks in the affected file will show up as 'missing' in an icheck(1) or fsck(1) of the filesystem.

Read and write permission is required on the specified file system device. The i-node 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 i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node 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 i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

icheck(1), fsck(1), ipatch(1)

RESTRICTIONS

If the file is open, clri is likely to be ineffective.

The clri command should be used only as a last resort, fsck provides a much safer method of accomplishing file system repairs.

NAME

cmp - compare two files

SYNOPSIS

cmp [-l] [-s] file1 file2

DESCRIPTION

The two files are compared. (If file1 is '-', the standard input is used.) Under default options, cmp makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:

-l Print the byte number (decimal) and the differing bytes (octal) for each difference.

-s Print nothing for differing files; return codes only.

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

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

COL(1)

NAME

col - filter reverse line feeds

SYNOPSIS

col [-bfx]

DESCRIPTION

Col reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ESC-7 in ASCII) and by forward and reverse half line feeds (ESC-9 and ESC-8). Col is particularly useful for filtering multicolumn output made with the '.rt' command of nroff and output resulting from use of the tbl(1) preprocessor.

Although col accepts half line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full line boundary. This treatment can be suppressed by the -f (fine) option; in this case the output from col may contain forward half line feeds (ESC-9), but will still never contain either kind of reverse line motion.

If the -b option is given, col assumes that 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.

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.

Col normally converts white space to tabs to shorten printing time. If the -x option is given, this conversion is suppressed.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by one of 789, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other nonprinting characters are ignored.

SEE ALSO

troff(1), tbl(1), greek(1)

RESTRICTIONS

Can't back up more than 128 lines.
No more than 800 characters, including backspaces, on a line.

NAME

comb - combine SCCS deltas

SYNOPSIS

comb [-o] [-s] [-psid] [-clist] files

DESCRIPTION

Comb generates a shell procedure (see sh(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named 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.

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.

-pSID The SCCS IDentification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

-clist A list (see get(1) for the syntax of a list) of deltas to be preserved. All other deltas are discarded.

-o For each get -e generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the -o keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

-s This argument causes comb to 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:

100 * (original - combined) / original

It is recommended that before any SCCS files are

COMB(1)

actually combined, one should use this option to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, comb will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB	The name of the reconstructed SCCS file.
comb?????	Temporary.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5). Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use help(1) for explanations.

RESTRICTIONS

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 - select or reject lines common to two sorted files

SYNOPSIS

comm [- [123]] file1 file2

DESCRIPTION

Comm reads file1 and file2, which should be ordered in ASCII collating sequence, and produces a three column output: lines only in file1; lines only in file2; and lines in both files. The filename '-' means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus comm -12 prints only the lines common to the two files; comm -23 prints only lines in the first file but not in the second; comm -123 is a no-op.

SEE ALSO

cmp(1), diff(1), uniq(1)

CP(1)

NAME

cp - copy

SYNOPSIS

cp file1 file2

cp file ... directory

DESCRIPTION

file1 is copied onto file2. The mode and owner of file2 are preserved if it already existed; the mode of the source file is used otherwise.

In the second form, one or more files are copied into the directory with their original file-names.

Cp refuses to copy a file onto itself.

SEE ALSO

cat(1), pr(1), mv(1)

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -o [acBv]

cpio -i [Bcdmrtuvfs6] [patterns]

cpio -p [adlmruv] directory

DESCRIPTION

Cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

Cpio -i (copy in) extracts files from the standard input which is assumed to be the product of a previous cpio -o. Only files with names that match patterns are selected. Patterns are given in the name-generating notation of sh(1). In patterns, meta-characters ?, *, and [...] match the slash / character. Multiple patterns may be specified and if no patterns are specified, the default for patterns is * (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

Cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination directory tree based upon the options described below.

The meanings of the available options are:

a Reset access times of input files after they have been copied.

B Input/output is to be blocked 5,120 bytes to the record (does not apply to the pass option; meaningful only with data directed to or from /dev/rmt?,/dev/rht?).

d Directories are to be created as needed.

c Write header information in ASCII character form for portability.

r Interactively rename files. If the user types a null line, the file is skipped.

t Print a table of contents of the input. No files are created.

u Copy unconditionally (normally, an older file will not replace a newer file with the same name).

v Verbose: causes a list of file names to be printed. When used with the t option, the table of contents looks like the output of an ls -l command (see ls(1)).

l Whenever possible, link files rather than copying them. Usable only with the -p option.

CPIO(1)

m Retain previous file modification time. This option is ineffective on directories that are being copied.
f Copy in all files except those in patterns.
s Swap bytes. Use only with the -i option.
S Swap halfwords. Use only with the -i option.
b Swap both bytes and halfwords. Use only with the -i option.
6 Process an old (i.e., UNIX System Sixth Edition format) file. Only useful with -i (copy in).

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/ht0

cd olddir
find . -depth -print | cpio -pdl newdir
```

The trivial case "find . -depth -print | cpio -oB >/dev/rmt0" can be handled more efficiently by:

```
find . -cpio /dev/rmt0
```

SEE ALSO

ar(1), find(1), cpio(5).

RESTRICTIONS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files.

NAME

csf, msf - simplify special file creation

SYNOPSIS

/etc/csf [-r] [-f #] [-c #] dn# [tty#] ... dn# [tty#]

cd /dev
msf devtype unit [tty#]

DESCRIPTION

The csf program creates all of the special files for a device with a single command. The optional '-r' specifies that the special files are to be removed instead of created.

The optional '-f #' argument causes only the special file for disk partition # to be created or removed. Multiple '-f' arguments may not be given.

The '-c #' argument specifies the MSCP disk controller number, zero is assumed if the '-c #' is omitted.

The dn# argument specifies the device name and unit number. The 'dn' is the ULTRIX-11 mnemonic for the device, see the ULTRIX-11 System Management Guide, Appendix C for a list of device mnemonics. For all devices, except DL and KL, # is the unit number. For disks and tapes, if the # is omitted the special files for all possible units will be created or removed. For all other devices # must be present. For the KL and DL devices, # is the number of units present.

The optional 'tty#' specifies the TTY line number associated with the first port on the device. This argument must be given as shown, i.e., the letters 'tty' followed by the line number. If the 'tty#' argument is omitted line zero is assumed. If both KL and DL devices are present, then their special files must be created with a single execution of the csf command, for example:

```
/etc/csf kl4 tty16 dl3 tty20
```

This is necessary because the starting minor device number for the DL depends on how many KL devices are present. Do not create the special file for the console terminal (KL0).

The msf command is a shell procedure that uses the /etc/csf program to create the special files for a device. The 'devtype' argument is the generic name of the device, i.e., rp06, ra80, tu16, lp11, dz11, etc. The 'devtype' takes the form 'devtype_cn' for disks that can be attached to multiple controllers, '_cn' is the controller number. For example, rp06_1 for an RP06 disk on the second RH11/RH70 MASSBUS disk controller, or rx50_2 for an RX50 on the third MSCP disk

controller. The 'unit' argument is the unit number. The 'tty#' is identical to the tty# used with the csf program, except that is must be present.

FILES

/dev/*

SEE ALSO

mknod(1)
ULTRIX-11 System Management Guide, Sections 1.4 and 2.6.

NAME

csh - a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [-cefinstvVxX] [arg ...]

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see History Substitutions), job control facilities (see Jobs) and a C-like syntax. So as to be able to use its job control facilities, users of csh must (and automatically do) use the new tty driver summarized in newtty(4) and fully described in tty(4). This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See stty(1) for details on setting options in the new tty driver. Login(1) normally switches to the new line discipline if the shell is specified as /bin/csh; otherwise it sets the terminal stop characters to be undefined. In this way, users of /bin/oldcsh may use the same shell without job control if that name is a link to /bin/csh.

An instance of csh begins by executing commands from the file '.cshrc' in the home directory of the invoker. If this is a login shell then it also executes commands from the file '.login' there. It is typical for users on crt's to put the command 'stty crt' in their .login file, and to also invoke tset(1) there.

In the normal case, the shell will then begin reading commands from the terminal, prompting with '% '. Processing of arguments and the use of the shell to process files containing command scripts will be described later.

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 command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file '.logout' in the users home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters '&' '|' ';' '<' '>' '(' ')' form separate words. If doubled in '&&', '||', '<<' or '>>' these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with '\'. A newline preceded by a '\' is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, ''', '`' or '"', form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of ''' or '"' characters a newline preceded by a '\' gives a true newline character.

When the shell's input is not a terminal, the character '#' introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by '\' and in quotations using '`', ''', and '"'.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by '|' characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ';', and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an '&'.

Any of the above may be placed in '(' ')' to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with '||' or '&&' indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See Expressions.)

Jobs

The shell associates a job with each pipeline. It keeps a table of current jobs, printed by the jobs command, and assigns them small integer numbers. When a job is started asynchronously with '&', the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the bg command, or run some other commands

and then eventually bring the job back into the foreground with the foreground command fg. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is another special key ^Y which does not generate a STOP signal until a program attempts to read(2) it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command 'stty tostop'. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character '%' introduces a job name. If you wish to refer to job number 1, you can name it as '%1'. Just naming a job brings it to the foreground; thus '%1' is a synonym for 'fg %1', bringing job 1 back into the foreground. Similarly saying '%1 &' resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous, thus '%ex' would normally restart a suspended ex(1) job, if there were only one suspended job whose name began with the string 'ex'. It is also possible to say '%?string' which specifies a job whose text contains string, if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a '+' and the previous job with a '-'. The abbreviation '%+' refers to the current job and '%-' refers to the previous job. For close analogy with the syntax of the history mechanism (described below), '%%' is also a synonym for the current job.

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable notify, the shell will notify you immediately of changes of status in background jobs. There is also a shell command notify which marks a single process so that its status changes will be immediately reported. By default notify marks the current process; simply say 'notify' after starting a background job to mark it.

CSH(1)

When you try to leave the shell while jobs are stopped, you will be warned that 'You have stopped jobs.' You may use the jobs command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character '!' and may begin anywhere in the input stream (with the proviso that they do not nest.) This '!' may be preceded by an '\' to prevent its special meaning; for convenience, a '!' is passed unchanged when it is followed by a blank, tab, newline, '=' or '('. (History substitutions also occur when an input line begins with '^'. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of which is controlled by the history variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the history command:

```
 9  write michael
10  ex write.c
11  cat oldwrite.c
12  diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the prompt by placing an '!' in the prompt string.

With the current event 13 we can refer to previous events by event number '!11', relatively as in '!-2' (referring to the

same event), by a prefix of a command word as in '!d' for event 12 or '!wri' for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially a redo.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of a input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

0	first (command) word
n	n'th argument
^	first argument, i.e. '1'
$	last argument
%	word matched by (immediately preceding) ?s? search
x-y	range of words
-y	abbreviates '0-y'
*	abbreviates '^-$', or nothing if only 1 word in event
x*	abbreviates 'x-$'
x-	like 'x*' but omitting word '$'

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '^', '$', '*' '-' or '%'. After the optional word designator can be placed a sequence of modifiers, each preceded by a ':'. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.xxx' component, leaving the root name.
e	Remove all but the extension '.xxx' part.
s/l/r/	Substitute l for r
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, for example, 'g&'.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a 'g' the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

CSH(1)

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of '/'; a '\' quotes the delimiter into the l and r strings. The character '&' in the right hand side is replaced by the text from the left. A '\' quotes '&' also. A null l uses the previous string either from a l or from a contextual scan string s in '!?s?'. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing '?' in a contextual scan.

A history reference may be given without an event specification, e.g. '!$'. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus '!?foo?^!$' gives the first and last arguments from the command matching '?foo?'.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a '^'. This is equivalent to '!:s^' providing a convenient shorthand for substitutions on the text of the previous line. Thus '^lb^lib' fixes the spelling of 'lib' in the previous command. Finally, a history substitution may be surrounded with '{' and '}' if necessary to insulate it from the characters which follow. Thus, after 'ls -ld ~paul' we might do '!{l}a' to do 'ls -ld ~paula', while '!la' would look for a command starting 'la'.

Quotations with ' and "

The quotation of strings by ''' and '"' can be used to prevent all or some of the remaining substitutions. Strings enclosed in ''' are prevented any further interpretation. Strings enclosed in '"' are yet variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see Command Substitution below) does a '"' quoted string yield parts of more than one word; ''' quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the alias and unalias commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words

replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for 'ls' is 'ls -l' the command 'ls /usr' would map to 'ls -l /usr', the argument list here being undisturbed. Similarly if the alias for 'lookup' was 'grep !^/etc/passwd' then 'lookup bill' would map to 'grep bill /etc/passwd'.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can 'alias print 'pr \!* | lpr'' to make a command which pr's its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the argv variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the set and unset commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the verbose variable is a toggle which causes command input to be echoed. The setting of this variable results from the -v command line option.

Other operations treat variables numerically. The '@' command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '$' characters. This expansion can be prevented by preceding the '$' with a '\' except within '"'s where it always occurs, and within '''s where it never occurs. Strings quoted by '`' are interpreted later (see Command

CSH(1)

substitution below) so '$' substitution does not occur there until later, if at all. A '$' is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in '"' or given the ':q' modifier the results of variable substitution may eventually be command and filename substituted. Within '"' a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the ':q' modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}

Are replaced by the words of the value of variable name, each separated by a blank. Braces insulate name from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.
If name is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}

May be used to select only some of the words from the value of name. The selector is subjected to '$' substitution and may consist of a single number or two numbers separated by a '-'. The first word of a variables value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '$#name'. The selector '*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
Gives the number of words in the variable. This is useful for later use in a '[selector]'.

$0
Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}
Equivalent to '$argv[number]'.

$*
Equivalent to '$argv[*]'.

The modifiers ':h', ':t', ':r', ':q' and ':x' may be applied to the substitutions above as may ':gh', ':gt' and ':gr'. If braces '{' '}' appear in the command form then the modifiers must appear within the braces. The current implementation allows only one ':' modifier on each '$' expansion.

The following substitutions may not be modified with ':' modifiers.

$?name
${?name}
Substitutes the string '1' if name is set, '0' if it is not.

$?0
Substitutes '1' if the current input filename is known, '0' if it is not.

$$
Substitute the (decimal) 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 shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This

CSH(1)

occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in '`'. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within '"'s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters '*', '?', '[' or '{' or begins with the character '~', then that word is a candidate for filename substitution, also known as 'globbing'. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters '*', '?' and '[' imply pattern matching, the characters '~' and '{' being more akin to abbreviations.

In matching filenames, the character '.' at the beginning of a filename or immediately following a '/', as well as the character '/' must be matched explicitly. The character '*' matches any string of characters, including the null string. The character '?' matches any single character. The sequence '[...]' matches any one of the characters enclosed. Within '[...]', a pair of characters separated by '-' matches any character lexically between the two.

The character '~' at the beginning of a filename is used to refer to home directories. Standing alone, i.e. '~' it expands to the invokers home directory as reflected in the value of the variable home. When followed by a name consisting of letters, digits and '-' characters the shell searches for a user with that name and substitutes their home directory; thus '~ken' might expand to '/usr/ken' and '~ken/chmach' to '/usr/ken/chmach'. If the character '~' is followed by a character other than a letter or '/' or appears not at the beginning of a word, it is left undisturbed.

The metanotation 'a{b,c,d}e' is a shorthand for 'abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus '~source/sl/{oldls,ls}.c' expands to '/usr/source/sl/oldls.c /usr/source/sl/ls.c' whether or not these files exist without any chance of error if the home directory for 'source' is '/usr/source'. Similarly '../{memo,*box}' might expand to '../memo ../box ../mbox'. (Note that 'memo' was not sorted with the results of matching '*box'.) As a special case '{', '}' and '{}' are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

< name
: Open file name (which is first variable, command and filename expanded) as the standard input.

<< word
: Read the shell input up to a line which is identical to word. Word is not subjected to variable, filename or command substitution, and each input line is compared to word before any substitutions are done on this input line. Unless a quoting '\', '"', ''' or '`' appears in word variable and command substitution is performed on the intervening lines, allowing '\' to quote '$', '\' and '`'. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name
: The file name is used as standard output. If the file does not exist then it is created; if the file exists, its is truncated, its previous contents being lost.

  If the variable noclobber is set, then the file must not exist or be a character special file (e.g. a terminal or '/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the '!' forms can be used and suppress this check.

  The forms involving '&' route the diagnostic output into the specified file as well as the standard output. Name is expanded in the same way as '<' input filenames are.

>> name
>>& name
>>! name
>>&! name
Uses file name as standard output like '>' but places output at the end of the file. If the variable noclobber is set, then it is an error for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is not modified to be the empty file '/dev/null'; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see Jobs above.)

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, exit, if, and while commands. The following operators are available:

|| && | ^ & == != =~ !~ <= >= < > << >> + -
* / % ! ~ ( )

Here the precedence increases to the right, '==' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '*' '/' and '%' being, in groups, at the same level. The '==' '!=' '=~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '=~' and '!~' are like '!=' and '==' except that the right hand side is a pattern (containing, e.g. '*'s, '?'s and instances of '[...]') against which the left hand operand is matched.

This reduces the need for use of the switch statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '-l name' where l is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable status examined.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The foreach, switch, and while statements, as well as the if-then-else form of the if statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in

this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias
alias name
alias name wordlist
The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified wordlist as the alias of name; wordlist is command and filename substituted. Name is not allowed to be alias or unalias.

alloc
Shows the amount of dynamic core in use, broken down into used and free core, and address of the last location in the heap. With an argument shows each used and free block on the internal dynamic memory chain indicating its address, size, and whether it is used or free. This is a debugging command and may not work in production versions of the shell; it requires a modified version of the system memory allocator.

bg
bg %job ...
Puts the current or specified jobs into the background, continuing them if they were stopped.

break
Causes execution to resume after the end of the nearest enclosing foreach or while. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a switch, resuming after the endsw.

case label:
A label in a switch statement as discussed below.

cd
cd name
chdir
chdir name
Change the shells working directory to directory name. If no argument is given then change to the home

directory of the user.
If name is not found as a subdirectory of the current directory (and does not begin with '/', './' or '../'), then each component of the variable cdpath is checked to see if it has a subdirectory name. Finally, if all else fails but name is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing while or foreach. The rest of the commands on the current line are executed.

default:
Labels the default case in a switch statement. The default should come after all case labels.

dirs
Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist
echo -n wordlist
The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the -n option is specified.

else
end
endif
endsw
See the description of the foreach, if, switch, and while statements below.

eval arg ...
(As in sh(1).) The arguments are read as input to the shell and the resulting command(s) executed. 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 using eval.

exec command
The specified command is executed in place of the current shell.

exit
exit(expr)
The shell exits either with the value of the status variable (first form) or with the value of the

specified expr (second form).

fg
fg %job ...
Brings the current or specified jobs into the foreground, continuing them if they were stopped.

foreach name (wordlist)
...
end
The variable name is successively set to each member of wordlist and the sequence of commands between this command and the matching end are executed. (Both foreach and end must appear alone on separate lines.)

The builtin command continue may be used to continue the loop prematurely and the builtin 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. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist
Like echo but no '\' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word
The specified word is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

hashstat
Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding exec's). An exec is attempted for each component of the path where the hash function indicates a possible hit, and in each component which does not begin with a '/'.

history
history n
history -r n
Displays the history event list; if n is given only the n most recent events are printed. The -r option reverses the order of printout to be most recent first rather than oldest first.

if (expr) command
If the specified expression evaluates true, then 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. Input/output 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 the specified expr is true then the commands to the first else are executed; else if expr2 is true then the commands to the second else are executed, etc. Any number of else-if pairs are possible; only one endif is needed. The else part is likewise optional. (The words else and endif must appear at the beginning of input lines; the if must appear alone on its input line or after an else.)

jobs
jobs -l
Lists the active jobs; given the -l options lists process id's in addition to the normal information.

kill %job
kill -sig %job ...
kill pid
kill -sig pid ...
kill -l
Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in /usr/include/signal.h, stripped of the prefix 'SIG'). The signal names are listed by 'kill -l'. There is no default, saying just '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 will be sent a CONT (continue) signal as well.

limit
limit resource
limit resource maximum-use
(Vax only.) Limits the consumption by the current process and each process it creates to not individually exceed maximum-use on the specified resource. If no

CSH(1)

maximum-use is given, then the current limit is printed; if no resource is given, then all limitations are given.

Resources controllable currently include cputime (the maximum number of cpu-seconds to be used by each process), filesize (the largest single file which can be created), datasize (the maximum growth of the data+stack region via sbrk(2) beyond the end of the program text), stacksize (the maximum size of the automatically-extended stack region), and coredumpsize (the size of the largest core dump that will be created).

The maximum-use may be given as a (floating point or integer) number followed by a scale factor. For all limits other than cputime the default scale is 'k' or 'kilobytes' (1024 bytes); a scale factor of 'm' or 'megabytes' may also be used. For cputime the default scaling is 'seconds', while 'm' for minutes or 'h' for hours, or a time of the form 'mm:ss' giving minutes and seconds may be used.

For both resource names and scale factors, unambiguous prefixes of the names suffice.

login
Terminate a login shell, replacing it with an instance of /bin/login. This is one way to log off, included for compatibility with sh(1).

logout
Terminate a login shell. Especially useful if ignoreeof is set.

newgrp
Changes the group identification of the caller; for details see newgrp(1). A new shell is executed by newgrp so that the shell state is lost.

nice
nice +number
nice command
nice +number command
The first form sets the nice for this shell to 4. The second form sets the nice to the given number. The final two forms run command at priority 4 and number respectively. The superuser may specify negative niceness by using 'nice -number ...'. Command is always executed in a sub-shell, and the restrictions place on commands in simple if statements apply.

nohup
nohup command
The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with '&' are effectively nohup'ed.

notify
notify %job ...
Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable notify is set.

onintr
onintr -
onintr label
Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form 'onintr -' causes all interrupts to be ignored. The final form causes the shell to execute a 'goto label' when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of onintr have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd
popd +n
Pops the directory stack, returning to the new top directory. With a argument '+n' discards the nth entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd
pushd name
pushd +n
With no arguments, pushd exchanges the top two elements of the directory stack. Given a name argument, pushd changes to the new directory (ala cd) and pushes the old current working directory (as in csw) onto the directory stack. With a numeric argument, rotates the nth argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash
Causes the internal hash table of the contents of the directories in the path variable to be recomputed. This is needed if new commands are added to directories in the path while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command
The specified command which is subject to the same restrictions as the command in the one line if statement above, is executed count times. I/O redirections occur exactly once, even if count is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)
The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets name to the null string. The third form sets name to the single word. The fourth form sets the index'th component of name to word; this component must already exist. The final form sets name to the list of words in wordlist. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value
Sets the value of environment variable name to be value, a single string. The most commonly used environment variable 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.

shift
shift variable
The members of argv are shifted to the left, discarding argv[1]. It is an error for argv not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name
The shell reads commands from name. Source commands may

be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a source at any level terminates all nested source commands. Input during source commands is never placed on the history list.

stop
stop %job ...
Stops the current or specified job which is executing in the background.

suspend
Causes the shell to stop 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(1).

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw
Each case 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, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command breaksw causes execution to continue after the endsw. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the endsw.

time
time command
With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the time variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask value
The file creation mask is displayed (first form) or set to the specified value (second form). The mask is

CSH(1)

given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be unaliased.

unhash
Use of the internal hash table to speed location of executed programs is disabled.

unlimit resource
unlimit
(Vax only.) Removes the limitation on resource. If no resource is specified, then all resource limitations are removed.

unset pattern
All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset *'; this has noticeably distasteful side-effects. It is not an error for nothing to be unset.

unsetenv pattern
Removes all variables whose name match the specified pattern from the environment. See also the setenv command above and printenv(1).

wait
All background jobs are waited for. It the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end
While the specified expression evaluates non-zero, the commands between the while and the matching end are evaluated. Break and continue may be used to terminate or continue the loop prematurely. (The while and end must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the foreach statement if the input is a terminal.

%job
Brings the specified job into the foreground.

%job &
Continues the specified job in the background.

@
@ name = expr
@ name[index] = expr
The first form prints the values of all the shell variables. The second form sets the specified name to the value of expr. If the expression contains '<', '>', '&' or '|' then at least this part of the expression must be placed within '(' ')'. The third form assigns the value of expr to the index'th argument of name. Both name and its index'th component must already exist.

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 which would otherwise be single words.

Special postfix '++' and '--' operators increment and decrement name respectively, i.e. '@ i++'.

Pre-defined and environment variables

The following variables have special meaning to the shell. Of these, argv, cwd, home, path, prompt, shell and status are always set by the shell. Except for cwd and status this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable user, TERM into term, and HOME into home, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file .cshrc as inferior csh processes will import the definition of path from the environment, and re-export it if you then change it. (It could be set once in the .login except that commands through net(1) would not see the definition.)

argv — Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. '$1' is replaced by '$argv[1]', etc.

cdpath — Gives a list of alternate directories searched to find subdirectories in chdir commands.

CSH(1)

cwd — The full pathname of the current directory.

echo — Set when the -x command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.

history — Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of history may run the shell out of memory. The last executed command is always saved on the history list.

home — The home directory of the invoker, initialized from the environment. The filename expansion of '~' refers to this variable.

ignoreeof — If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.

mail — The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.

If the first word of the value of mail is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says 'New mail in name' when there is mail in the file name.

noclobber — As described in the section on Input/output, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that '>>' redirections refer to existing files.

noglob — If set, filename expansion is inhibited. This is most useful in shell scripts which

are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch — If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. 'echo [' still gives an error.

notify — If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a prompt.

path — Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no path variable then only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the superuser the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the -c nor the -t option will normally hash the contents of the directories in the path variable after reading .cshrc, and each time the path variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the rehash or the commands may not be found.

prompt — The string which is printed before each command is read from an interactive terminal input. If a '!' appears in the string it will be replaced by the current event number unless a preceding '\' is given. Default is '% ', or '# ' for the superuser.

shell — The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of Non-builtin Command Execution below.) Initialized to the (system-dependent) home of the shell.

status — The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail

CSH(1)

return exit status '1', all other builtin commands set status '0'.

time — Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose — Set by the -v command line option, causes the words of each command to be printed after history substitution.

Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via execve(2). Each word in the variable path names a directory from which the shell will attempt to execute the command. If it is given neither a -c nor a -t option, the shell will hash the names in these directories into an internal table so that it will only try an exec in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via unhash), or if the shell was given a -c or -t argument, and in any case for each directory component of path which does not begin with a '/', the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus '(cd ; pwd) ; pwd' prints the home directory; leaving you where you were (printing this after the home directory), while 'cd ; pwd' leaves you in the home directory. Parenthesized commands are most often used to prevent chdir from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands an a new shell is spawned to read it.

If there is an alias for shell then the words of the alias will be prepended to the argument list to form the shell command. The first word of the alias should be the full path name of the shell (e.g. '$shell'). Note that this is a special, late occurring, case of alias substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell is '-' then this is a login shell. The flag arguments are interpreted as follows:

-c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in argv.

-e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

-f The shell will start faster, because it will neither search for nor execute commands from the file '.cshrc' in the invokers home directory.

-i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

-n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

-s Command input is taken from the standard input.

-t A single line of input is read and executed. A '\' may be used to escape the newline at the end of this line and continue onto another line.

-v Causes the verbose variable to be set, with the effect that command input is echoed after history substitution.

-x Causes the echo variable to be set, so that commands are echoed immediately before execution.

-V Causes the verbose variable to be set even before '.cshrc' is executed.

-X Is to -x as -V is to -v.

After processing of flag arguments if arguments remain but none of the -c, -i, -s, or -t options was given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by '$0'. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a 'standard' shell if the first character of a script is not a '#', i.e. if the script does not start with a comment. Remaining arguments initialize the variable

argv.

Signal handling

The shell normally ignores quit signals. Jobs running detached (either by '&' or the bg or %... & commands) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by onintr. Login shells catch the terminate signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file '.logout'.

AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp of I.I.A.S.A, Laxenburg, Austria, with different syntax than that used now.

FILES

~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after '.cshrc' at login.
~/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a '#'.
/tmp/sh*	Temporary file for '<<'.
/etc/passwd	Source of home directories for '~name'.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters (5120 characters on most PDP11 systems). 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.

SEE ALSO

oldcsh(1), sh(1), access(2), execve(2), fork(2), killpg(2), pipe(2), sigsys(2), umask(2), wait(2), exec(3), jobs(3), sigset(3), tty(4), a.out(5), environ(5), 'An introduction to the C shell'

RESTRICTIONS

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 (i.e. wrong) as the job may have changed directories internally.

Shell builtin 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 will then immediately execute 'c'. This is especially noticeable if this expansion results from an alias. It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. '(a ; b ; c)'.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by '?', are not placed in the history list. Control structure 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. All and more than one ':' modifier should be allowed on '$' substitutions.

CTAGS(1)

NAME

ctags - create a tags file

SYNOPSIS

ctags [-BFatuwvx] name ...

DESCRIPTION

Ctags makes a tags file for ex(1) from the specified C, Pascal and Fortran 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 blanks or tabs. Using the tags file, ex can quickly find these object name definitions.

If the -x flag is given, ctags produces 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.

If the -v flag is given, an index of the form expected by vgrind(1) is produced on the standard output. 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. Sample use:

```
ctags -v files | sort -f > index
vgrind -x index
```

Files whose name ends in .c or .h are assumed to be C source files and are searched for C routine and macro definitions. 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.

Other options are:

-B use backward searching patters (?...?).

-F use forward searching patterns (/.../) (default).

-a append to tags file.

-t create tags for typedefs.

-u causing the specified files to be updated 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.)

-w suppressing warning diagnostics.

The tag main is treated specially in C programs. The tag formed is created by prepending M to the name of the file, 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.

FILES

tags output tags file

SEE ALSO

ex(1), vi(1)

AUTHOR

Ken Arnold; FORTRAN added by Jim Kleckner; Bill Joy added Pascal and -x, replacing cxref; C typedefs added by Ed Pelegri-Llopart.

RESTRICTIONS

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.

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

tip, cu - connect to a remote system

SYNOPSIS

tip [-v] [-speed] system-name
tip [-v] [-speed] phone-number
cu phone-number [-t] [-s speed] [-a acu] [-l line] [-#]

DESCRIPTION

Tip and cu establish a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote cpu. It goes without saying that you must have a login on the machine (or equivalent) to which you wish to connect. The preferred interface is tip. The cu interface is included for those people attached to the 'call UNIX' command of version 7. This manual page describes only tip.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde ('~') appearing as the first character of a line is an escape signal; the following are recognized:

~^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 will return you to tip).

~> Copy file from local to remote. Tip prompts for the name of a local file to transmit.

~< Copy file from remote to local. Tip prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

~p from [to]
Send a file to a remote UNIX host. The put command causes the remote UNIX system to run the command string 'cat > 'to' ', while tip sends it the 'from' file. If the 'to' file isn't specified the 'from' file name is used. This command is actually a UNIX specific version of the '~>' command.

~t from [to]
Take a file from a remote UNIX host. As in the put command the 'to' file defaults to the 'from' file

name if it isn't 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 UNIX process. The command string sent to the local UNIX system is processed by the shell.

~# Send a BREAK to the remote system. For systems which don't support the necessary ioctl call the break is simulated by a sequence of line speed changes and DEL characters.

~s Set a variable (see the discussion below).

~^Z Stop tip (only available with job control).

~? Get a summary of the tilde escapes

Tip uses the file /etc/remote to find how to reach a particular system and to find out how it should operate while talking to the system; refer to remote(5) for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, e.g. 'tip -300 mds'.

When tip establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in /etc/remote.

When tip prompts for an argument (e.g. 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, will abort the dialogue and return you to the remote machine.

Tip guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by uucp(1C).

During file transfers tip provides a running count of the number of lines transferred. When using the ~> and ~< commands, the 'eofread' and 'eofwrite' variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, 'echocheck' may be set to indicate tip should synchronize with the remote system on the echo of each transmitted character.

When tip must dial a phone number to connect to a system it will print various messages indicating its actions. Tip supports the DEC DN-11 and Racal-Vadic 831 auto-call-units; the DEC DF02 and DF03, Ventel 212+, Racal-Vadic 3451, and Bizcomp 1031 and 1032 integral call unit/modems.

VARIABLES

Tip maintains a set of variables which control its operation. Some of these variable 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 set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a '?' to the end. For example '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 concatenating 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 file .tiprc in one's home directory). The -v option causes tip to display the sets as they are made. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values.

beautify
(bool) Discard unprintable characters when a session is being scripted; abbreviated be.

baudrate
(num) The baud rate at which the connection was established; abbreviated ba.

dialtimeout
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated dial.

echocheck
(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is off.

eofread
(str) The set of characters which signify and end-of-transmission during a ~< file transfer command; abbreviated eofr.

eofwrite
(str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated eofw.

eol
(str) The set of characters which indicate an end-of-line. Tip will recognize escape characters only after an end-of-line.

escape
(char) The command prefix (escape) character; abbreviated es; default value is '~'.

exceptions
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated ex; default value is '\t\n\f\b'.

force
(char) The character used to force literal data transmission; abbreviated fo; default value is '^P'.

framesize
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated fr.

host
(str) The name of the host to which you are connected; abbreviated ho.

prompt
(char) The character which indicates and end-of-line on the remote host; abbreviated pr; default value is '\n'. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on recipt of this character.

raise
(bool) Upper case mapping mode; abbreviated ra; default value is off. When this mode is enabled, all lower case letters will be mapped to upper case by tip for transmission to the remote machine.

raisechar
(char) The input character used to toggle upper case mapping mode; abbreviated rc; default value is '^A'.

record
(str) The name of the file in which a session script is recorded; abbreviated rec; default value is 'tip.record'.

script
(bool) Session scripting mode; abbreviated sc; default is off. When script is true, 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 betwee 040 and 0177). The variable exceptions is used to indicate characters which are an exception to the normal beautification rules.

tabexpand
(bool) Expand tabs to spaces during file transfers; abbreviated tab; default value is false. Each tab is expanded to 8 spaces.

verbose
(bool) Verbose mode; abbreviated verb; default is true. When verbose mode is enabled, tip prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

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.

FILES

/etc/remote	global system descriptions
/etc/phones	global phone number data base
${REMOTE}	private system descriptions
${PHONES}	private phone numbers
~/.tiprc	initialization file.
/usr/spool/uucp/LCK..*	lock file to avoid conflicts with uucp

DIAGNOSTICS
Diagnostics are, hopefully, self explanatory.

SEE ALSO
remote(5), phones(5)

RESTRICTIONS
The full set of variables is undocumented and should, probably, be paired down.

CU_V7(1C)

NAME

cu_v7 - call UNIX (original V7 version)

SYNOPSIS

cu_v7 telno [-t] [-s speed] [-a acu] [-l line]

DESCRIPTION

Cu v7 is the original version of CU from UNIX V7, used with the DN11 and the Bell 801 auto call unit. Cu v7 is obsolete and is not supported by the ULTRIX-11 system.

Cu v7 calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of text files. Telno is the telephone number, with minus signs at appropriate places for delays. The -t flag is used to dial out to a terminal. Speed gives the transmission speed (110, 134, 150, 300, 1200); 300 is the default value.

The -a and -l values may be used to specify pathnames for the ACU and communications line devices. They can be used to override the following built-in choices:

-a /dev/cua0 -l /dev/cul0

After making the connection, cu v7 runs as two processes: the send process reads the standard input and passes most of it to the remote system; the receive process reads from the remote system and passes most data to the standard output. Lines beginning with '~' have special meanings.

The send process interprets the following:

~.	terminate the conversation.
~EOT	terminate the conversation
~<file	send the contents of file to the remote system, as though typed at the terminal.
~!	invoke an interactive shell on the local system.
~!cmd ...	run the command on the local system (via sh -c).
~$cmd ...	run the command locally and send its output to the remote system.
~%take from [to]	copy file 'from' (on the remote system) to file 'to' on the local system. If 'to' is omitted, the 'from' name is used both places.

~%put from [to] copy file 'from' (on local system) to file 'to' on remote system. If 'to' is omitted, the 'from' name is used both places.

~~... send the line '~...'.

The receive process handles output diversions of the following form:

```
~>[>][:]file
zero or more lines to be written to file
~>
```

In any case, output is diverted (or appended, if '>>' used) to the file. If ':' is used, the diversion is silent, i.e., it is written only to the file. If ':' is omitted, output is written both to the file and to the standard output. The trailing '~>' terminates the diversion.

The use of ~%put requires stty and cat on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of ~%take requires the existence of echo and tee on the remote system. Also, stty tabs mode is required on the remote system if tabs are to be copied without expansion.

FILES

/dev/cua0
/dev/cul0
/dev/null

SEE ALSO

dn(4), tty(4)

DIAGNOSTICS

Exit code is zero for normal exit, nonzero (various values) otherwise.

RESTRICTIONS

The syntax is unique.

NAME

cu_v7m, custat_v7m - call UNIX (V7M version)

SYNOPSIS

cu_v7m telno [-t] [-n] [-m] [-s speed] [-a acu] [-l line]
custat_v7m

DESCRIPTION

Cu_v7m is the version of the cu command supplied with the ULTRIX-11 V1.0 software. Cu_v7m remains a part of the ULTRIX-11 system, but its use is not recommended. Digital recommends use of the tip utility in place of cu, refer to tip(1c) for details.

Cu_v7m calls up another UNIX system, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of text files. Cu_v7m establishes the system to system connection by calling the remote system via DF02-AC or DF03-AC auto call modem, or over a direct link with the remote system. Telno is the telephone number, with an equal sign at the appropriate place if an intermediate dial tone is expected. The -t flag is used to access the remote system over a direct link instead of an auto call unit. Speed gives the transmission speed (110, 150, 300, 1200, 2400, 4800, 9600); the default value is 9600 for direct connects (-t) and 1200 for auto dial connections.

The -a and -l values may be used to specify pathnames for the ACU and communications line devices. They can be used to override the following built-in choices:

-a /dev/cua0 -l /dev/cul0

The -n option, where n is a single digit, changes the last character of the ACU and communications line to n. It is an abbreviation for -a /dev/cuan -l /dev/culn.

The -m option is used in conjunction with the -t flag when the direct link involves modems. This option forces data terminal ready (DTR) to be asserted by the communications line device driver.

After making the connection, cu_v7m runs as two processes: the send process reads the standard input and passes most of it to the remote system; the receive process reads from the remote system and passes most data to the standard output. Lines beginning with '~' have special meanings, which are acted upon only after the receipt of a newline.

The send process interprets the following:

~.	terminate the conversation.
~<file	send the contents of file to the remote system, as though typed at the terminal.
~#	sends a break (cycles speeds on auto bit rate lines).
~!	invoke an interactive shell on the local system.
~!cmd ...	run the command on the local system (via sh -c).
~$cmd ...	run the command locally and send its output to the remote system.
~%take from [to]	copy file 'from' (on the remote system) to file 'to' on the local system. If 'to' is omitted, the 'from' name is used both places.
~%put from [to]	copy file 'from' (on local system) to file 'to' on remote system. If 'to' is omitted, the 'from' name is used both places.
~:	during an output diversion, this toggles whether the operation of cu v7m will be silent, i.e., whether information received from the remote system will be written to the standard output. This allows a "progress report" during long transfers.
~~...	send the line '~...'.

Both the send and receive processes handle output diversions of the following form:

```
~>[>][:]file
zero or more lines to be written to file
~>
```

In any case, output is diverted (or appended, if '>>' used) to the file. If ':' is used, the diversion is silent, i.e., it is written only to the file. If ':' is omitted, output is written both to the file and to the standard output. The trailing '~>' terminates the diversion.

The use of ~%put requires stty and cat on the remote side. It also requires that the current erase and kill characters

CU_V7M(1C)

on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of ~%take requires the existence of echo and tee on the remote system. Also, stty tabs mode is required on the remote system if tabs are to be copied without expansion.

The custat v7m command shows the status of all auto call units and communications lines. This enables the users of cu to more easily select an available ACU or line.

FILES

/dev/cua[0-7]
/dev/cul[0-7]
/dev/null
/usr/spool/uucp/LCK..cu[al][0-7] - lock files
/usr/adm/culog - log file

SEE ALSO

tty(4)
ULTRIX-11 System Management Guide, Section 4.8

DIAGNOSTICS

Exit code is zero for normal exit, nonzero (various values) otherwise.

RESTRICTIONS

The syntax is unique.
A ~%put cannot be aborted by typing the interrupt character, ~. will stop the transfer and disconnect the line.
When typing lines that begin with '~', erase and kill processing do not function properly, i.e., the characters are deleted but still remain on the screen.
The lack of proper flow control causes file transfers at speeds of greater than 1200 BPS to be unreliable, depending on the amount of activity on the local and remote systems.

NAME

cut - cut out selected fields of each line of a file

SYNOPSIS

cut -clist [filel file2 ...]
cut -flist [-dchar] [-s] [filel file2 ...]

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 fixed length, i.e., character positions as on a punched card (-c option), or the length can vary from line to line and be marked with a field delimiter character like tab (-f option). Cut can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

list — A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges as in the -o option of nroff/troff for page ranges; e.g., 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field).

-clist — The list following -c (no space) specifies character positions (e.g., -c1-72 would pass the first 72 characters of each line).

-flist — The list following -f is a list of fields assumed to be separated in the file by a delimiter character (see -d); e.g. , -f1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless -s is specified.

-dchar — The character following -d is the field delimiter (-f option only). Default is tab. Space or other characters with special meaning to the shell must be quoted.

-s — Suppresses lines with no delimiter characters in case of -f option. Unless specified, lines with no delimiters will be passed through untouched.

Either the -c or -f option must be specified.

HINTS

Use grep(1) to make horizontal 'cuts' (by context) through a file, or paste(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use cut and paste.

EXAMPLES

```
cut -d: -f1,5 /etc/passwd
                        mapping of user IDs to names

name=`who am i | cut -f1 -d" "`
                        to set name to current login name.
```

DIAGNOSTICS

line too long
: A line can have no more than 511 characters or fields.

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.

no fields
: The list is empty.

SEE ALSO

grep(1), paste(1).

NAME

date - print and set the date

SYNOPSIS

date [-u] [yymmddhhmm [.ss]]

DESCRIPTION

If no argument is given, the current date and time are printed. If an argument is given, the current date is set. 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 is optional and is the seconds. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The year, month and day may be omitted, the current values being the defaults. The system operates in GMT. Date takes care of the conversion to and from local standard and daylight time. The -u flag causes date to set and print the date and time in GMT without converting to local time.

FILES

/usr/adm/wtmp to record time-setting

SEE ALSO

utmp(5)

DIAGNOSTICS

'No permission' if you aren't the super-user and you try to change the date; 'bad conversion' if the date set is syntactically incorrect.

NAME

dc - desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. 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, then from the standard input. The following constructions are recognized:

number

The value of the number is pushed on 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. Numbers 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; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

sx The top of the stack is popped and stored into a register named x, where x may be any character. If the s is capitalized, x is treated as a stack and the value is pushed on it.

lx The value in register x is pushed on the stack. The register x is not altered. All registers start with zero value. If the l is capitalized, register x is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. P interprets the top of the stack as an ascii string, removes it, and prints it.

f All values on the stack and in registers are printed.

q exits the program. If executing a string, the recursion level is popped by two. If q is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x treats the top element of the stack as a character string and executes it as a string of dc commands.

X replaces the number on the top of the stack with its scale factor.

[...]
puts the bracketed ascii string onto the top of the stack.

<x >x =x
The top two elements of the stack are popped and compared. Register x is executed if they obey the stated relation.

v replaces 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.

! interprets the rest of the line as a UNIX command.

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input. I pushes the input base on the top of the stack.

o The top value on the stack is popped and used as the number radix for further output.

O pushes 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 The stack level is pushed onto the stack.

Z replaces the number on the top of the stack with its length.

? A line of input is taken from the input source (usually the terminal) and executed.

; : are used by bc for array operations.

An example that prints the first ten values of n! is

```
[lal+dsa*plal0>y]sy
0sal
lyx
```

SEE ALSO

bc(1), which is a preprocessor for dc providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

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.

NAME

dcheck - file system directory consistency check

SYNOPSIS

dcheck [-i numbers] filesystem

DESCRIPTION

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The -i flag is followed by 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.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

FILES

There are no default file systems, they must be specified.

SEE ALSO

icheck(1), filsys(5), clri(1), ncheck(1), fsck(1)

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.

RESTRICTIONS

Since dcheck is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

For the most part icheck (1) has been superseded by fsck (1).

NAME

dcopy - copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [options] inputfs outputfs

DESCRIPTION

Dcopy copies file system inputfs to outputfs. Inputfs is the existing file system; outputfs is an appropriately sized file system, to hold the reorganized result. Inputfs must be the raw device and outputfs must be the block device. Dcopy must be run on unmounted file systems (in the case of the root file system, copy to a new pack). With no arguments, dcopy copies files from inputfs compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap.

Options:

-sX supply device information for creating an optimal organization of blocks in a file. The forms of X are the same as the -s option of fsck(1M). Dcopy will attempt to use the values from the output filesystem if the parameters given in this option are not reasonable.

-an place the files not accessed in n days after the free blocks of the destination file system (default for n is 7). If no a is specified then no movement occurs.

-d leave order of directory entries as is (default is to move sub-directories to the beginning of directories).

-v currently reports how many files were processed, and how big the source and destination freelists are.

-n do not ask before copying file systems. Default is to ask for confirmation before proceeding.

-f*fsize*[:*isize*]
specify the *outputfs* file system and inode list sizes (in blocks). If not given, the values from the *outputfs* are used. *Dcopy* catches interrupts and quits and reports on its progress. To terminate *dcopy*, send a quit signal and *dcopy* will no longer catch interrupts or quits. *Dcopy* also attempts to modify its command line arguments so its progress can be monitored with *ps*(1).

SEE ALSO
fsck(1M), mkfs(1M), ps(1).

DD(1)

NAME

dd - convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the 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.

option	values
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=n	input block size n bytes (default 512)
obs=n	output block size (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 files from (tape) input
seek=n	seek n records from beginning of output file before copying
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
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
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with k, b or w to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by x to indicate a product.

Cbs is used only if ascii or ebcdic conversion is specified. In the former case cbs characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, 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.

For example, to read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file x:

```
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.

To skip over a file before copying from magnetic tape do

```
(dd of=/dev/null; dd of=x) </dev/rmt0
```

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

f+p records in(out): numbers of full and partial records read(written)

RESTRICTIONS

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.

Newlines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

CAUTION

Image copying disks using dd requires careful selection of the block size (bs, ibs, obs). Using the default block size of 512 bytes (1b) will always work, however this is not very efficient. In most cases the optimum block size is the number of 512 byte sectors per track. For example: to copy an RL02 disk to another RL02 disk use 'bs=20b', or to copy an RP06 to an RM03 use 'ibs=22b obs=32b'. If the size of the input disk is larger that the size of the output disk a portion of the last record will not be transferred. An disk image copy using dd will always terminate with the message 'No such device or address', this indicates that the end of the disk has been reached.

DELTA(1)

NAME

delta - make a delta (change) to an SCCS file

SYNOPSIS

delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

DESCRIPTION

Delta is used to permanently introduce 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 keyletters specified and flags (see admin(1)) that may be present in the SCCS file (see -m and -y keyletters below).

Keyletter arguments apply independently to each named file.

-rSID — Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding gets for editing (get -e) on the same SCCS file were done by the same person (login name). The SID value specified with the -r keyletter can be either the SID specified on the get command line or the SID to be made as reported by the get command (see get(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

-s — Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n — Specifies retention of the edited g-file (normally removed at completion of delta processing).

-g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

-m[*mrlist*] If the SCCS file has the v flag set (see *admin*(1)) then 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 keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that 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).

-y[*comment*] Arbitrary text used 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 new-line character terminates the comment text.

-p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form ?-file are explained in the *Source*

DELTA(1)

Code Control System User's Guide. 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/bdiff	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 will cause 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 keyletters must also be present. Omission of these keyletters causes an error to occur.

SEE ALSO

admin(1), bdiff(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use help(1) for explanations.

NAME

deroff - remove nroff, troff, tbl and eqn constructs

SYNOPSIS

deroff [-w] file ...

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.

If the -w flag is given, the output is a word list, one 'word' (string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed) per line, and all other characters ignored. Otherwise, the output follows the original, with the deletions mentioned above.

SEE ALSO

troff(1), eqn(1), tbl(1)

RESTRICTIONS

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.

DF(1M)

NAME

df - disk free

SYNOPSIS

df [filesystem] ...

DESCRIPTION

Df prints out the number of free blocks available on the filesystems. If no file system is specified, the free space on all of the normally mounted file systems is printed.

FILES

Default file system names obtained from the /etc/fstab file.

SEE ALSO

icheck(1), fsck(1), fstab(5)

NAME

diff - differential file comparator

SYNOPSIS

diff [-efbh] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If file1 (file2) is '-', the standard input is used. If file1 (file2) is a directory, then a file in that directory whose file-name is the same as the file-name of file2 (file1) is used. The normal output contains lines of these forms:

n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4

These lines resemble ed commands to convert file1 into file2. The numbers after the letters pertain to file2. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert file2 into file1. As in ed, 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 '>'.

The -b option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The -e option produces a script of a, c and d commands for the editor ed, which will recreate file2 from file1. The -f option produces a similar script, not useful with ed, in the opposite order. 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
```

Except in rare circumstances, diff finds a smallest sufficient set of file differences.

Option -h does 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 -e and -f are unavailable with -h.

FILES
/tmp/d?????
/usr/lib/diffh for -h

SEE ALSO
cmp(1), comm(1), ed(1)

DIAGNOSTICS
Exit status is 0 for no differences, 1 for some, 2 for trouble.

RESTRICTIONS
Editing scripts produced under the -e or -f option are naive about creating lines consisting of a single '.'.

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

diff3 [-ex3] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	file1 is different
====2	file2 is different
====3	file3 is different

The type of change suffered in converting a given range of a given file to some other is 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.

Under the -e option, diff3 publishes a script for the editor ed that will incorporate into file1 all changes between file2 and file3, i.e. the changes that normally would be flagged ==== and ====3. Option -x (-3) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to 'file1'.

```
(cat script; echo '1,$p') | ed - file1
```

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(1)

RESTRICTIONS

Text lines that consist of a single '.' will defeat -e.
Files longer than 64K bytes won't work.

DU(1)

NAME

du - summarize disk usage

SYNOPSIS

du [-s] [-a] [name ...]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each specified directory or file name. If name is missing, '.' is used.

The optional argument -s causes only the grand total to be given. The optional argument -a causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

A file which has two links to it is only counted once.

RESTRICTIONS

Non-directories given as arguments (not under -a option) are not listed.
If there are too many distinct linked files, du counts the excess files multiply.

NAME

dump - incremental file system dump

SYNOPSIS

dump [key [argument ...] filesystem]

DESCRIPTION

Dump copies to the dump media, usually magtape, all files changed after a certain date in the filesystem. The key specifies the date and other options about the dump. Key consists of characters from the set 0123456789fusdmy.

Before beginning the the dump startup confirmation is requested. This helps prevent accidental overwriting of file systems due to improper dump file specifications.

f Place the dump on the next argument file instead of the tape.

u If the dump completes successfully, write the date of the beginning of the dump on file '/etc/ddate'. This file records a separate date for each filesystem and each dump level.

0-9 This number is the 'dump level'. All files modified since the last date stored in the file '/etc/ddate' for the same filesystem at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the option 0 causes the entire filesystem to be dumped.

s The size of the dump tape is specified in feet. The number of feet is taken from the next argument. When the specified size is reached, the dump will wait for reels to be changed. The default size is 2300 feet.

d The density of the tape, expressed in BPI, is taken from the next argument. This is used in calculating the amount of tape used per write. The default is 1600.

m Micro/pdp-11, dump to RX50 diskettes instead of tape. The f option must also be specified when m is used. For example: dump 0umf /dev/rrx2 /dev/rrd0

y Suppress the start of dump confirmation query, allows for non-interactive use of dump.

If no arguments are given, the key is assumed to be 9u and a default file system is dumped to the default tape.

Now a short suggestion on how perform dumps. Start with a full level 0 dump

dump 0u

Next, periodic level 9 dumps should be made on an exponential progression of tapes. (Sometimes called Tower of Hanoi - 1 2 1 3 1 2 1 4 ... tape 1 used every other time, tape 2 used every fourth, tape 3 used every eighth, etc.)

dump 9u

When the level 9 incremental approaches a full tape (about 78000 blocks at 1600 BPI blocked 20), a level 1 dump should be made.

dump 1u

After this, the exponential series should progress as uninterrupted. These level 9 dumps are based on the level 1 dump which is based on the level 0 full dump. This progression of levels of dump can be carried as far as desired.

FILES

/dev/null - default file system to be dumped
/dev/rht0 - default dump device
/etc/ddate - record dump dates of filesystem/level.

SEE ALSO

restor(1), dump(5), dumpdir(1)
ULTRIX-11 System Management Guide, Sections 4.2 and 5.1.4

DIAGNOSTICS

If the dump requires more than one tape, it will ask you to change tapes. Reply with a new-line when this has been done.

Unless the y option is specified dump asks for confirmation, as follows:

Last chance before writing on /dev/r??##
Continue <y or n> ?

A read error on the file system will result in an error message, but the dump will continue.

Unrecoverable write errors on the dump device cause the dump of the current volume to be aborted. The dump program will ask if the current volume is to be rewritten. Answer yes to continue the dump with the current volume or no to abort the entire dump.

RESTRICTIONS

Sizes are based on 1600 BPI blocked tape. The raw magtape device has to be used to approach these densities.

NAME

dumpdir - print the names of files on a dump tape

SYNOPSIS

dumpdir [f filename]

DESCRIPTION

Dumpdir is used to read magtapes dumped with the dump command and list the names and inode numbers of all the files and directories on the tape.

The f option causes filename as the name of the tape instead of the default.

FILES

/dev/rht0 - default tape
rst*

SEE ALSO

dump(5), dump(1), restor(1)

DIAGNOSTICS

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

RESTRICTIONS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, dumpdir doesn't use it.

NAME

echo - echo arguments

SYNOPSIS

echo [-n] [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a newline on the standard output. If the flag -n is used, no newline is added to the output.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, do 'echo ... 1>&2'.

ED(1)

NAME

ed - text editor

SYNOPSIS

ed [-] [name]

red [-] [name]

DESCRIPTION

Ed is the standard text editor.

If a name argument is given, ed simulates an e command (see below) on the named file; that is to say, the file is read into ed's buffer so that it can be edited. The optional - suppresses the printing of character counts by e, r, and w commands, of diagnostics from e and q commands, and of the ! prompt after a !shell command.

Ed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a w (write) command is given. The copy of the text being edited resides in a temporary file called the buffer.

Red is a restricted version of ed. It will only allow editing of files in the current directory. It prohibits executing shell commands via !shell command. Attempts to bypass these restrictions result in an error message (restricted shell).

Commands to ed have a simple and regular structure: zero or more addresses followed by a single character command, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to the buffer. While ed is accepting text, it is said to be in input mode. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period '.' alone at the beginning of a line.

Ed supports a limited form of regular expression notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

The regular expressions allowed by ed are constructed as follows:

The following one-character regular expressions match a single character:

1.1 An ordinary character (not one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash (\) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, except when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the beginning of an entire regular expression (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (currency symbol), which is special at the end of an entire regular expression (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire regular expression, which is special for that regular expression (for example, see how slash (/) is used in the g command, below.)

1.3 A period (.) is a one-character regular expression that matches any character except new-line.

1.4 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 circumflex (^), 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); e.g., []a-f] matches either a right square bracket (]) or one of the letters a through f

ED(1)

inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct regular expressions from one-character regular expressions:

2.1 A one-character regular expression is a regular expression that matches whatever the one-character regular expression matches.

2.2 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.

2.3 A one-character regular expression followed by \{m\}, \{m,\}, or \{m,n\} is a regular expression that matches a range of occurrences of the one-character 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.

2.4 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.5 A regular expression enclosed between the character sequences \(and \) is a regular expression that matches whatever the unadorned regular expression matches.

2.6 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 sub-expression 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.

Finally, an entire regular expression may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A circumflex (^) at the beginning of an entire

regular expression constrains that regular expression to match an initial segment of a line.

3.2 A currency symbol ($) at the end of an entire regular expression constrains that regular expression to match a final segment of a line.

Regular expressions are used in addresses to specify lines and in one command (see s below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

The construction entire regular expression$ constrains the entire regular expression to match the entire line.

The null regular expression (e.g., //) is equivalent to the last regular expression encountered. See also the last paragraph before FILES below.

To understand addressing in ed it is necessary to know that at any time there is a current line. Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

1. The character '.' addresses the current line.

2. The character '$' addresses the last line of the buffer.

3. A decimal number n addresses the n-th line of the buffer.

4. 'x addresses the line marked with the name x, which must be a lowercase letter. Lines are marked with the k command described below.

5. A regular expression enclosed in slashes '/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer. See also the last paragraph before FILES below.

6. A regular expression enclosed in queries '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the

search wraps around to the end of the buffer. See also the last paragraph before FILES below.

7. An address followed by a plus sign '+' or a minus sign '-' followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.

8. If an address begins with '+' or '-' the addition or subtraction is taken with respect to the current line; e.g. '-5' is understood to mean '.-5'.

9. If an address ends with '+' or '-', then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character in addresses is entirely equivalent to -.) Moreover, trailing '+' and '-' characters have a cumulative effect, so '--' refers to the current line less 2.

10. For convenience, a comma (,) stands for the address pair 1,$, while a semicolon (;) stands for the pair .,$.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient ones are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma ','. They may also be separated by a semicolon ';'. In this case the current line '.' is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches ('/', '?'). The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of ed commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

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 suffixed by l, n or p, in which case the current line is either listed, numbered or printed, respectively, as

discussed below under the l, n and p commands.

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer. See also the last paragraph before FILES below.

(.,.)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(.,.)d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no 'filename' is given, the currently-remembered file name, if any, is used (see the f command). The number of characters read is typed; 'filename' is remembered for possible use as a default file name in subsequent e, r, and w commands. If 'filename' is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose output is to be read. Such a shell command is not remembered as the current file name. See also DIAGNOSTICS below.

E filename

This command is the same as e, except that no diagnostic results when no w has been given since the last buffer alteration.

f filename

The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1,$)g/regular expression/command list

ED(1)

In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '.' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with '\'. A, i, and c commands and associated input are permitted; the '.' terminating input mode may be omitted if it would be on the last line of the command list. The g, G, v, and V commands are not permitted in the command list. See also RESTRICTIONS and the last paragraph before FILES below.

(1,$)G/ regular expression /

In the interactive Global command, the first step is to mark every line that matches the given regular expression. Then, for every such line, that line is printed, '.' is changed to that line, and any one command (other than one of the a, c, i, g, G, v, and V commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an '&' causes the re-execution of the most recent command executed within the current invocation of G. Note that the commands input as part of the execution of the G command may address and affect any lines in the buffer. The G command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The help command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The Help command causes ed to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The H command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

This command inserts the given text before the addressed line. '.' is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the a command only in the placement of the text. Address 0 is not legal 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
This command joins the addressed lines into a single line; intermediate newlines simply disappear. '.' is left at the resulting line.

(.)kx
The mark command marks the addressed line with name x, which must be a lowercase letter. The address 'x then addresses this line; '.' is unchanged.

(.,.)l
The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The l command may be placed on the same line after any non-i/o command.

(.,.)ma
The move command repositions the addressed line(s) after the line addressed by a. Address 0 is legal for a and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address a falls within the range of moved lines; '.' is left at the last line moved.

(.,.)n
The number command prints the addressed lines, preceding each line by its line number and a tab character; '.' is left at the last line printed. The number command may placed on the same line after any non-i/o command.

(.,.)p
The print command prints the addressed lines. '.' is left at the last line printed. The p command may be placed on the same line after any non-i/o command.

P The editor will prompt with an asterisk (*) for all subsequent commands. The P command alternately turns this mode on and off; it is initially off.

q The quit command causes ed to exit. No automatic write of a file is done.

Q This command is the same as q, except that no diagnostic results when no w has been given since the last buffer alteration.

($)r filename
The read command reads in the given file after the

addressed line. If no 'filename' is given, the remembered file name, if any, is used (see e and f commands). The file name is remembered if there was no remembered file name already. Address 0 is legal for r and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '.' is left at the last line read in from the file. If 'filename' is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is not remembered as the current file name.

(.1,.)s/regular expression/replacement/
or,
(.,.)s/regular expression/replacement/g

The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, if the global replacement indicator 'g' appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of '/' to delimit the regular expression and the replacement. '.' is left at the last line substituted. See also the last paragraph before FILES below.

An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of '&' in this context may be suppressed by preceding it by '\'. The characters '\n' where n is a digit, are replaced by the text matched by the n-th regular subexpression enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, n is determined by counting occurrences of '\(' starting from the left.

Lines may be split by substituting new-line characters into them. The new-line in the replacement string must be escaped by preceding it by '\'. Such substitution cannot be done as part of a g, G, v, or V command list.

(.,.)ta
This command acts just like the m command, except that a copy of the addressed lines is placed after address a (which may be 0). '.' is left on the last line of the

copy.

u The undo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent a, c, d, g, i, j, m, r, s, t, v, G, or V command.

(1,$)v/ regular expression / 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 regular expression.

(1,$)V/ regular expression /
This command is the same as the interactive global command G except that the lines that are marked during the first step are those that do not match the regular expression.

(1,$)w filename
The write command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your umask setting (see sh(1)) dictates otherwise. The currently remembered 'filename' is not changed unless filename is the very first file name mentioned since ed was invoked. If no 'filename' is given, the currently remembered 'filename', if any, is used (see e and f commands); '.' is unchanged. If the command is successful, the number of characters written is typed. If filename is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose standard input is the addressed lines. Such a shell command is not remembered as the current file name.

(1,$)W filename
This command is the same as w, except that the addressed lines are appended to the file.

($)= The line number of the addressed line is typed. '.' is unchanged by this command.

!shell command
The remainder of the line after the ! is sent to the UNIX System shell (sh(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded

line is echoed; '.' is unchanged.

(.+1)<newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '.+1p'; it is useful for stepping through text.

If an interrupt signal (ASCII DEL) is sent, ed prints a '?' and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the temporary file. The limit on the number of lines depends on the amount of core: each line takes 1 word.

When reading a file, ed discards ASCII NUL characters and all characters after the last new-line. Files (e.g., a.out) that contain characters not in the ASCII set (bit 8 on) cannot be edited by ed.

If the closing delimiter of a regular expression or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may 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?
```

FILES

```
/tmp/e*
ed.hup     work is saved here if terminal hangs up
```

SEE ALSO

grep(1), sed(1), sh(1), stty(1)
B. W. Kernighan, A Tutorial Introduction to the ED Text Editor
B. W. Kernighan, Advanced editing on UNIX

DIAGNOSTICS

'?name' for inaccessible file; '?' for errors in commands; '?TMP' for temporary file overflow.
(use the help and Help commands for detailed explanations).
To protect against throwing away valuable work, a q or e command is considered to be in error, unless a w has occurred since the last buffer change. A second q or e will be obeyed regardless.

RESTRICTIONS

The l command mishandles DEL.
A ! command cannot be subject to a g or a v command.

The ! command and the ! escape from the e, r, and w commands cannot be used if the the editor is invoked from a restricted shell (see sh(1)).
Because 0 is an illegal address for a w command, it is not possible to create an empty file with ed.
Characters are masked to 7 bits on input.

NAME

ed - text editor

SYNOPSIS

ed [-] [name]

DESCRIPTION

Ed is the standard text editor.

If a name argument is given, ed simulates an e command (see below) on the named file; that is to say, the file is read into ed's buffer so that it can be edited. The optional - suppresses the printing of character counts by e, r, and w commands.

Ed operates on a copy of any file it is editing; changes made in the copy have no effect on the file until a w (write) command is given. The copy of the text being edited resides in a temporary file called the buffer.

Commands to ed have a simple and regular structure: zero or more addresses followed by a single character command, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to the buffer. While ed is accepting text, it is said to be in input mode. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period '.' alone at the beginning of a line.

Ed supports a limited form of regular expression notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be matched by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

1. Any character except a special character matches itself. Special characters are the regular expression delimiter plus \[. and sometimes ^*$.

2. A . matches any character.

3. A \ followed by any character except a digit or () matches that character.

4. A nonempty string s bracketed [s] (or [^s]) matches any character in (or not in) s. In s, \ has no special meaning, and] may only appear as the first letter. A

substring a-b, with a and b in ascending ASCII order, stands for the inclusive range of ASCII characters.

5. A regular expression of form 1-4 followed by * matches a sequence of 0 or more matches of the regular expression.

6. A regular expression, x, of form 1-8, bracketed \(x\) matches what x matches.

7. A \ followed by a digit n matches a copy of the string that the bracketed regular expression beginning with the nth \(matched.

8. A regular expression of form 1-8, x, followed by a regular expression of form 1-7, y matches a match for x followed by a match for y, with the x match being as long as possible while still permitting a y match.

9. A regular expression of form 1-8 preceded by ^ (or followed by $), is constrained to matches that begin at the left (or end at the right) end of a line.

10. A regular expression of form 1-9 picks out the longest among the leftmost matches in a line.

11. An empty regular expression stands for a copy of the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see s below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

To understand addressing in ed it is necessary to know that at any time there is a current line. Generally speaking, the current line is the last line affected by a command; however, the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows.

1. The character '.' addresses the current line.

2. The character '$' addresses the last line of the buffer.

3. A decimal number n addresses the n-th line of the buffer.

ED_V7(1)

4. ''x' addresses the line marked with the name x, which must be a lower-case letter. Lines are marked with the k command described below.

5. A regular expression enclosed in slashes '/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer.

6. A regular expression enclosed in queries '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer.

7. An address followed by a plus sign '+' or a minus sign '-' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.

8. If an address begins with '+' or '-' the addition or subtraction is taken with respect to the current line; e.g. '-5' is understood to mean '.-5'.

9. If an address ends with '+' or '-', then 1 is added (resp. subtracted). As a consequence of this rule and rule 8, the address '-' refers to the line before the current line. Moreover, trailing '+' and '-' characters have cumulative effect, so '--' refers to the current line less 2.

10. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is equivalent to '-'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma ','. They may also be separated by a semicolon ';'. In this case the current line '.' is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches ('/', '?'). The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of ed commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

As mentioned, it is generally illegal for more than one command to appear on a line. However, most commands may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed respectively in the way discussed below.

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.

(., .)c
<text>
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the line preceding the deleted lines.

(., .)d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in. '.' is set to the last line of the buffer. The number of characters read is typed. 'filename' is remembered for possible use as a default file name in a subsequent r or w command. If 'filename' is missing, the remembered name is used.

E filename

This command is the same as e, except that no diagnostic results when no w has been given since the last buffer alteration.

f filename

The filename command prints the currently remembered file name. If 'filename' is given, the currently

remembered file name is changed to 'filename'.

(1,$)g/regular expression/command list
In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '.' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with '\'. A, i, and c commands and associated input are permitted; the '.' terminating input mode may be omitted if it would be on the last line of the command list. The commands g and v are not permitted in the command list.

(.)i

<text>
.
This command inserts the given text before the addressed line. '.' is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the a command only in the placement of the text.

(., .+1)j
This command joins the addressed lines into a single line; intermediate newlines simply disappear. '.' is left at the resulting line.

(.)kx
The mark command marks the addressed line with name x, which must be a lower-case letter. The address form ''x' then addresses this line.

(., .)l
The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The l command may be placed on the same line after any non-i/o command.

(., .)ma
The move command repositions the addressed lines after the line addressed by a. The last of the moved lines becomes the current line.

(., .)p
The print command prints the addressed lines. '.' is left at the last line printed. The p command may be placed on the same line after any non-i/o command.

(., .)P
This command is a synonym for p.

q The quit command causes ed to exit. No automatic write of a file is done.

Q This command is the same as q, except that no diagnostic results when no w has been given since the last buffer alteration.

($)r filename
The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see e and f commands). The file name is remembered if there was no remembered file name already. Address '0' is legal for r and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '.' is left at the last line read in from the file.

(., .)s/regular expression/replacement/ or,
(., .)s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, if the global replacement indicator 'g' appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of '/' to delimit the regular expression and the replacement. '.' is left at the last line substituted.

An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of '&' in this context may be suppressed by preceding it by '\'. The characters '\n' where n is a digit, are replaced by the text matched by the n-th regular subexpression enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, n is determined by counting occurrences of '\(' starting from the left.

Lines may be split by substituting new-line characters into them. The new-line in the replacement string must be escaped by preceding it by '\'.

ED_V7(1)

(., .)ta
This command acts just like the m command, except that a copy of the addressed lines is placed after address a (which may be 0). '.' is left on the last line of the copy.

(., .)u
The undo command restores the preceding contents of the current line, which must be the last line in which a substitution was made.

(1, $)v/regular expression/command list
This command is the same as the global command g except that the command list is executed g with '.' initially set to every line except those matching the regular expression.

(1, $)w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writable by everyone). The file name is remembered if there was no remembered file name already. If no file name is given, the remembered file name, if any, is used (see e and f commands). '.' is unchanged. If the command is successful, the number of characters written is printed.

(1,$)W filename
This command is the same as w, except that the addressed lines are appended to the file.

($)= The line number of the addressed line is typed. '.' is unchanged by this command.

!<shell command>
The remainder of the line after the '!' is sent to sh(1) to be interpreted as a command. '.' is unchanged.

(.+1)<newline>
An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '.+1p'; it is useful for stepping through text.

If an interrupt signal (ASCII DEL) is sent, ed prints a '?' and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the temporary file. The limit on the number of lines depends on the amount of core: each line takes 1 word.

When reading a file, ed discards ASCII NUL characters and all characters after the last newline. It refuses to read files containing non-ASCII characters.

FILES

/tmp/e*
ed.hup: work is saved here if terminal hangs up

SEE ALSO

B. W. Kernighan, A Tutorial Introduction to the ED Text Editor
B. W. Kernighan, Advanced editing on UNIX
sed(1)

DIAGNOSTICS

'?name' for inaccessible file; '?' for errors in commands; '?TMP' for temporary file overflow.

To protect against throwing away valuable work, a q or e command is considered to be in error, unless a w has occurred since the last buffer change. A second q or e will be obeyed regardless.

RESTRICTIONS

The l command mishandles DEL.
A ! command cannot be subject to a g command.
Because 0 is an illegal address for a w command, it is not possible to create an empty file with ed.

EQN (1)

NAME

eqn, neqn, checkeq — typeset mathematics

SYNOPSIS

eqn [**−dxy**] [**−pn**] [**−sn**] [**−fn**] [file] ...
checkeq [file] ...

DESCRIPTION

Eqn is a troff(1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, *neqn* on terminals. Usage is almost always

```
eqn file ... | troff
neqn file ... | nroff
```

If no files are specified, these programs reads 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. Delimiters may be set to characters *x* and *y* with the command-line argument **−dxy** or (more commonly) with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

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 $\underline{\ddot{y}} = \underline{n}$, and *x vec* = *y dyad* is $\overrightarrow{x} = \overleftrightarrow{y}$.

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*.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **−p***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* (∞) 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 \(bs (⧵) 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)
B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics—User's Guide*
J. F. Ossanna, *NROFF/TROFF User's Manual*

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in 'bold "12.3"'.

NAME

ex, edit - text editor

SYNOPSIS

ex [-] [-v] [-r] [+command] [-l] name ...
edit [ex options]

DESCRIPTION

Ex is the root of a family of editors: edit, ex and vi. Ex is a superset of ed, with the most notable extension being a display editing facility. Display based editing is the focus of vi.

If you have not used ed, or are a casual user, you will find that the editor edit is convenient for you. It avoids some of the complexities of ex used mostly by systems programmers and persons very familiar with ed.

If you have a CRT terminal, you may wish to use a display based editor; in this case see vi(1), which is a command which focuses on the display editing portion of ex.

DOCUMENTATION

The document Edit: A tutorial provides a comprehensive introduction to edit assuming no previous knowledge of computers or the UNIX system.

The Ex Reference Manual - Version 3.5 is a comprehensive and complete manual for the command mode features of ex, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of ex see the editing documents written by Brian Kernighan for the editor ed; the material in the introductory and advanced documents works also with ex.

An Introduction to Display Editing with Vi introduces the display editor vi and provides reference material on vi. All of these documents can be found in volume 2c of the Programmer's Manual. In addition, the Vi Quick Reference card summarizes the commands of vi in a useful, functional way, and is useful with the Introduction.

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
/tmp/Exnnnnn	editor temporary
/tmp/Rxnnnnn	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), grep(1), vi(1), termcap(5), environ(5)

AUTHOR

Originally written by William Joy
Mark Horton has maintained the editor since version 2.7, adding macros, support for many unusual terminals, and other features such as word abbreviation mode.

RESTRICTIONS

The undo command causes all marks to be lost on lines changed and then restored 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 don't 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.

NAME

expr - evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expr | expr
yields the first expr if it is neither null nor '0', otherwise yields the second expr.

expr & expr
yields the first expr if neither expr is null or '0', otherwise yields '0'.

expr relop expr
where relop is one of < <= = != >= >, yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both expr are integers, otherwise lexicographic.

expr + expr
expr - expr
addition or subtraction of the arguments.

expr * expr
expr / expr
expr % expr
multiplication, division, or remainder of the arguments.

expr : expr
The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of ed(1). The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

(expr)
parentheses for grouping.

Examples:

To add 1 to the Shell variable a:

```
a=`expr $a + 1`
```

To find the filename part (least significant part) of the pathname stored in variable a, which may or may not contain '/':

```
expr $a : '.*/\(.*\)' '|' $a
```

Note the quoted Shell metacharacters.

SEE ALSO

ed(1), sh(1), test(1)

DIAGNOSTICS

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.

NAME

f77 - Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the UNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by f77.

In the same way, arguments whose names end with '.c' or '.s' are taken to be C or assembly source programs and are compiled or assembled, producing a '.o' file.

The following options have the same meaning as in cc(1). See ld(1) for load-time options.

-c Suppress loading and produce '.o' files for each source file.

-p Prepare object files for profiling, see prof(1).

-O Invoke an object-code optimizer.

-S Compile the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

-f Use a floating point interpreter (for PDP11's that lack 11/70-style floating point).

-o output
Name the final output file output instead of 'a.out'.

-V Produce code suitable for using in overlaid programs. This is a default flag.

-V7 Turn off the -V flag. This is provided for backwards compatibility with older versions of f77, but in general should not be used.

The following options are peculiar to f77.

-onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

-u Make the default type of a variable 'undefined' rather than using the default Fortran rules.

-C Compile code to check that subscripts are within declared array bounds.

-w Suppress all warning messages. If the option is '-w66', only Fortran 66 compatibility warnings are suppressed.

-F Apply EFL and Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to '.f', but do not compile.

-m Apply the M4 preprocessor to each '.r' or '.e' file before transforming it with the Ratfor or EFL preprocessor.

-Ex Use the string x as an EFL option in processing '.e' files.

-Rx Use the string x as a Ratfor option in processing '.r' files.

-N[qxscn]nnn
Make static tables in the compiler bigger. The compiler will complain if it overflows its tables and suggest you apply one or more of these flags. These flags have the following meanings:

q Maximum number of equivalenced variables. Default is 150.

x Maximum number of external names (common block names, subroutine and function names). Default is 200.

s Maximum number of statement numbers. Default is 201.

c Maximum depth of nesting for control statements (e.g. DO loops). Default is 20.

n Maximum number of identifiers. Default is 401.

Programs do not always require the full default space assigned to them. Accordingly, one of these default values may be reduced in order to accommodate an increased value for some other flag (see table below). In the case of much larger programs, static table space is at a premium. Therefore, experimentation with different values for these flags is sometimes required so that the compiler will not complain.

Storage costs associated with these optional flags are:

flag	default value	cost ea.	default cost
s	201	8 bytes	1608 bytes
q	150	12 bytes	1800 bytes
x	200	20 bytes	4000 bytes
c	20	18 bytes	360 bytes
n	401	4 bytes	1604 bytes

-T[12alFM]file
Use alternate programs for various passes of compilation.

1file
Use file for the back end of the compiler, instead of /lib/c1.

2file
Use file as the optimizer, instead of /lib/c2.

afile
Use file as the assembler, instead of /bin/as.

lfile
Use file as the loader, instead of /bin/ld.

Ffile
Use file as the footname (startup code), instead of /lib/crt0.

Mfile
Use file as the macro processor with the -m flag, instead of /usr/bin/m4.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name 'a.out'.

FILES

file.[fresc]	input file
file.o	object file
a.out	loaded output
/usr/lib/f77pass1	compiler
/lib/c1	pass 2
/lib/c2	optional optimizer
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library, see section 3

SEE ALSO

S. I. Feldman, P. J. Weinberger, A Portable Fortran 77 Compiler
prof(1), cc(1), ld(1)

DIAGNOSTICS

The diagnostics produced by f77 itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

RESTRICTIONS

The Fortran 66 subset of the language has been exercised extensively; the newer features have not.

NAME

factor, primes - factor a number, generate large primes

SYNOPSIS

factor [number]

primes

DESCRIPTION

When factor is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 256 (about 7.2e16) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If factor is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to sqrt(n) and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 1014 on a PDP11.

When primes is invoked, it waits for a number to be typed in. If you type in a positive number less than 256 it will print all primes greater than or equal to this number.

DIAGNOSTICS

'Ouch.' for input out of range or for garbage input.

NAME

file - determine file type

SYNOPSIS

file file ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ascii, file examines the first 512 bytes and tries to guess its language.

RESTRICTIONS

It often makes mistakes. In particular it often suggests that command files are C programs.

NAME

find - find files

SYNOPSIS

find pathname-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each pathname in the pathname-list (i.e., one or more pathnames) seeking files that match a boolean expression written in the primaries given below. 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.

-name filename
: True if the filename argument matches the current file name. Normal Shell argument syntax may be used if escaped (watch out for '[', '?' and '*').

-perm onum
: True if the file permission flags exactly match the octal number onum (see chmod(1)). If onum is prefixed by a minus sign, more flag bits (017777, see stat(2)) become significant and the flags are compared: (flags&onum)==onum.

-type c
: True if the type of the file is c, where c is b, c, d or f for block special file, character special file, directory or plain file.

-links n
: True if the file has n links.

-user uname
: True if the file belongs to the user uname (login name or numeric user ID).

-group gname
: True if the file belongs to group gname (group name or numeric group ID).

-size n
: True if the file is n blocks long (512 bytes per block).

-inum n
: True if the file has inode number n.

-atime n
: True if the file has been accessed in n days.

-mtime n
: True if the file has been modified in n days.

-exec command
: True if the executed command returns a zero value as exit status. The end of the 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; causes the current pathname to be printed.

-newer file
True if the current file has been modified more recently than the argument file.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

2) The negation of a primary ('!' is the unary not operator).

3) Concatenation of primaries (the and operation is implied by the juxtaposition of two primaries).

4) Alternation of primaries ('-o' is the or operator).

EXAMPLE
To remove all files named 'a.out' or '*.o' that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

FILES
/etc/passwd
/etc/group

SEE ALSO
sh(1), test(1), filsys(5)

RESTRICTIONS
The syntax is painful.

NAME

finger - user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default finger lists the login name, full name, terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX user. (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.)

A longer format also exists and is used by finger whenever a list of peoples names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file .plan in their home directory, and the project on which they are working from the file .project also in the home directory.

Finger options include:

-b Slightly briefer version (long format).

-f Suppress the printing of the header line (short format).

-h Suppress printing of the .project files.

-i Quick list (-q) with idle times.

-l Force long output format.

-p Suppress printing of the .plan files.

-q Quick list (similar to who(1)).

-s Force short output format.

-w Narrow format output (short format).

FILES

/etc/utmp	who file
/etc/passwd	for users names, offices, phones, directories, and shells
~/.plan	plans
~/.project	projects

SEE ALSO
who(1)

AUTHOR
Earl T. Cohen

RESTRICTIONS
Only the first line of the .project file is printed.

FPSIM(1M)

NAME

fpsim - report or change the status of floating point simulation

SYNOPSIS

/etc/fpsim [mode]

DESCRIPTION

Fpsim without any arguments will report the current status of the kernel floating point simulator, it is either enabled, disabled, or not configured in. If mode is specified it must be either onoroff, which will respectively enable or disable the simulator, provided that the simulator was configured into the current system. The new state is then reported. Only the super user is allowed to change the status of the simulator.

SEE ALSO

fpsim(2)

RESTRICTIONS

This command is only temporary, It is intended only for use if the kernel floating point simulator should blow up. Once it has been more thoroughly tested and verified there will no longer be a need to be able to turn it off on a running system.

NAME

from - who is my mail from?

SYNTAX

from [-f [mailbox]] [-s sender]

DESCRIPTION

From prints out the mail header lines in a mailbox file to show you who your mail is from. The -f option causes from to examine the mail header lines in your mbox (or the specified file). If the -s option is given, then only headers for mail sent by sender are printed.

FILES

/usr/spool/mail/*

SEE ALSO

mail(1)

FSCK(1M)

NAME

fsck - file system consistency check and interactive repair

SYNOPSIS

fsck [option] ... [filesystem] ...

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for the named filesystems. If a file system is consistent then the number of files. number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. Most corrections lose data; all losses are reported. The default action for each correction is to wait for the operator to respond 'yes' or 'no'. Without write permission fsck defaults to -n action.

These options are recognized:

-y Assume a yes response to all questions.

-n Assume a no response to all questions.

-sX Ignore the actual free list and (unconditionally) construct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done, or extreme care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The free list is created with optimal interleaving according to the specification X:

-s3 optimal for RP03 on PDP11/45 CPU
-s4 optimal for RP04, RP05, RP06 on PDP11/70 CPU
-sc:s space free blocks s blocks apart in cylinders of c blocks each.

If X is not given, the values used when the filesystem was created are used. If these values were not specified, then c=400, s=9 is assumed. The '-s' without X is recommended when rebuilding free lists on ULTRIX-11 file systems, because mkfs (1) saves the optimum interleave factors in the super-block when the file system is created.

-SX Conditionally reconstruct the free list. This option is like -sX except that the free list is rebuilt only if there were no discrepancies discovered in the file system. It is useful for forcing free list

reorganization on uncontaminated file systems. -S forces -n.

-t If fsck cannot obtain enough memory to keep its tables, it uses a scratch files. If the -t option is specified, the file named in the next argument is used as the scratch file. Without the -t option, fsck prompts if it needs a scratch file. The file should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when fsck completes.

If no filesystems 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. Size checks:
   Incorrect number of blocks in file.
   Directory size not a multiple of 16 bytes.

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 than 65536 inodes.
   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. The only restriction is that the directory "lost+found" must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making "lost+found",

copying a number of files to the directory, and then removing them (before fsck is executed).

Checking the raw device is almost always faster.

When checking the root file system the block device should be used. Fsck will force a reboot if any repairs were done on the root file system.

FILES

/etc/fstab - default file system check list

SEE ALSO

dcheck(1), icheck(1), filsys(5), fstab(5)
ULTRIX-11 System Management Guide, Sections 4.1, 4.6, and 5.1.3

RESTRICTIONS

Inode numbers for . and .. in each directory should be checked for validity.
The -b option of icheck(1) should be available.

NAME

fsdb - file system debugger

SYNOPSIS

/etc/fsdb special [-]

DESCRIPTION

Fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking fsdb with the optional - argument or by the use of the O symbol. (Fsdb reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by fsdb are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,-	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
" ="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the p symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The f symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the f symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A .B or .D is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
s0	high byte of file size
s1	low word of file size

a#	data block numbers (0 - 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.
2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first 512 bytes of the file are then printed in ASCII.
1b.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with nm if the first character is alphabetic.

SEE ALSO

fsck(1), dir(5), filsys(5), ipatch(1)

NAME

get - get a version of an SCCS file

SYNOPSIS

get [-rSID] [-ccutoff] [-ilist] [-xlist] [-aseq-no.] [-k] [-e] [-l[p]] [-p] [-m] [-n] [-s] [-b] [-g] [-t] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with -. The arguments may be specified in any order, but all keyletter arguments 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).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-rSID The SCCS IDentification 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 keyletter is also used), as a function of the SID specified.

-ccutoff 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: "-c77/2/2 9:22:25". Note that this

implies that one may use the %E% and %U% identification keywords (see below) for nested gets within, say the input to a send(1C) command:

```
~!get "-c%E% %U%" s.file
```

-e Indicates that the get is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of delta(1). The -e keyletter used in a get for a particular version (SID) of the SCCS file prevents further gets for editing on the same SID until delta is executed or the j (joint edit) flag is set in the SCCS file (see admin(1)). Concurrent use of get -e for different SIDs is always allowed.

If the g-file generated by get with an -e keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the get command with the -k keyletter in place of the -e keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see admin(1)) are enforced when the -e keyletter is used.

-b Used with the -e keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter 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.

-ilist 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 Identification 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.

-xlist A list of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the -i keyletter for the list format.

GET(1)

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The -k keyletter is implied by the -e keyletter.

-l[p] Causes a delta summary to be written into an *l-file*. If -lp is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the -s keyletter is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m Causes each text line retrieved from the SCCS file to be preceded by 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 Causes each generated text line to be preceded 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 keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the -m keyletter generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

-t Used to access the most recently created ('top') delta in a given release (e.g., -r1), or release and level (e.g., -r1.2).

-a*seq-no*. The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the -r and -a keyletters are specified, the -a keyletter is used. Care should be taken when using the -a

keyletter in conjunction with the -e keyletter, as the SID of the delta to be created may not be what one expects. The -r keyletter can be used with the -a and -e keyletters to control the naming of the SID of the delta to be created.

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 keyletter 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 file name is printed (preceded by a new-line) before it is processed. If the -i keyletter is used included deltas are listed following the notation 'Included'; if the -x keyletter is used, excluded deltas are listed following the notation 'Excluded'.

TABLE 1. Determination of SCCS Identification String

SID* Specified	-b Keyletter 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 in release > R and R exists	hR.mL**	hR.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 (i.e., maximum branch number plus one) of level L within release R'. Note that if the 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.

*** This is used to force creation of the first delta in a new release.

Successor.

† The -b keyletter 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, then 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(1) strings for UNIX program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing what(1) strings for non-UNIX program files. %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 keyletter 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 keyletter 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 keyletter 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 wasn't applied and ignored;
   * if the delta wasn't applied and wasn't 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 keyletter).
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 delta.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The p-file is used to pass information resulting from a get with an -e keyletter along to delta. Its contents are also used to prevent a subsequent execution of get with an -e keyletter 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 keyletter argument if it was present, followed by a blank and the -x keyletter argument 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 (i.e., 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

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use _help_(1) for explanations.

RESTRICTIONS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the -e keyletter is used.

NAME

graph - draw a graph

SYNOPSIS

graph [option] ...

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(1) 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.

The following options are recognized, each as a separate argument.

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument 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 Character string given by next argument is default label for each point.

-g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

-l Next argument is label for graph.

-m Next argument is 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]
If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.

-y [l]
Similarly for y.

-h Next argument is fraction of space for height.

-w Similarly for width.

-r Next argument is fraction of space to move right before plotting.

-u Similarly to move up before plotting.

-t Transpose horizontal and vertical axes. (Option -x now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the -s option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO
spline(1), plot(1)

RESTRICTIONS
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.

GREP(1)

NAME

grep, egrep, fgrep - search a file for a pattern

SYNOPSIS

grep [option] ... expression [file] ...

egrep [option] ... [expression] [file] ...

fgrep [option] ... [strings] [file]

DESCRIPTION

Commands of the grep family search the input files (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output; unless the -h flag is used, the file name is shown if there is more than one input file.

Grep patterns are limited regular expressions in the style of ed(1); it uses a compact nondeterministic algorithm. Egrep patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. Fgrep patterns are fixed strings; it is fast and compact.

The following options are recognized.

-v All lines but those matching are printed.

-c Only a count of matching lines is printed.

-l The names of files with matching lines are listed (once) separated by newlines.

-n Each line is preceded by its line number in the file.

-b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

-s No output is produced, only status.

-h Do not print filename headers with output lines.

-y Lower case letters in the pattern will also match upper case letters in the input (grep only).

-e expression
Same as a simple expression argument, but useful when the expression begins with a -.

-f file
The regular expression (egrep) or string list (fgrep) is taken from the file.

-x (Exact) only lines matched in their entirety are printed (fgrep only).

Care should be taken when using the characters $ * [^ | ? ' " ( ) and \ in the expression as they are also meaningful to the Shell. It is safest to enclose the entire expression argument in single quotes ' '.

Fgrep searches for lines that contain one of the (newline-separated) strings.

Egrep accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character matches that character.

The character ^ ($) matches the beginning (end) of a line.

A . matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the string. Ranges of ASCII character codes may be abbreviated as in 'a-z0-9'. A] may occur only as the first character of the string. A literal - must be placed where it can't be mistaken as a range indicator.

A regular expression followed by * (+, ?) matches a sequence of 0 or more (1 or more, 0 or 1) matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

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 [] then *+? then concatenation then | and newline.

SEE ALSO

ed(1), sed(1), sh(1)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

RESTRICTIONS

Ideally there should be only one grep, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters; longer lines are truncated.

NAME

help - ask for help

SYNOPSIS

help [args]

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 (e.g., ge6, for message 6 from the get command).

type 2 — Does not contain numerics (as a command, such as get)

type 3 — Is all numeric (e.g., 212)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try 'help stuck'.

FILES

/usr/lib/help — directory containing files of message text.

DIAGNOSTICS

Use help(1) for explanations.

HOSTNAME(1)

NAME

hostname - set or print name of current host system

SYNOPSIS

hostname [nameofhost]

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.

SEE ALSO

ghostname(2)

NAME

icheck - file system storage consistency check

SYNOPSIS

icheck [-s] [-b numbers] filesystem

DESCRIPTION

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; i.e. not in any file nor in the free list.

The -s option causes icheck to ignore the actual free list and reconstruct a new one by rewriting the super-block 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 super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The -s option causes the normal output reports to be suppressed.

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

There is no default file system name.

SEE ALSO

dcheck(1), ncheck(1), filsys(5), clri(1), fsck(1)

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.

RESTRICTIONS

Since icheck is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.
It believes even preposterous super-blocks and consequently can get core images.

The fsck (1) command provides a more comprehensive file system check.

NAME

iostat - report I/O statistics

SYNOPSIS

iostat [option] ... [drive] ... [interval [count]]

DESCRIPTION

Iostat delves into the system and reports certain statistics kept about input-output activity. Information is kept about up to nine different types of disks (HP, HM, HJ, HK, RP, RL, RK, HS, RA) and about typewriters. The RA disks include the RX50 floppy disk (RX), the RD51 winchester disk (RD), and the RC25 disk.

For each disk drive, I/O completions and number of words transferred are counted; for typewriters collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk drive is examined and a tally is made if the disk drive is active. The processor state is also examined, this tally goes into one of four categories, depending on whether the system is executing in user mode, in 'nice' (background) user mode, in system mode, or idle.

The iostat reports are for all types of activity, seeks as well as data transfers, on all drives that have had any I/O activity since the system was booted, inactive and nonexistent drives are ignored.

The optional drive argument allows the reports to be limited to a specified subset of the available drives. Up to six drive names, of the form; hp0, hm1, rl3, rp4, rd1, ra2, etc., may be specified. Reports will be generated for only those drives which exist and have been active.

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. If count is given, then interval must also be specified.

With no option argument iostat reports for each disk the number of transfers per minute, the milliseconds per average seek, and the milliseconds per data transfer exclusive of seek time. It also gives the percentage of time the system has spend in each of the four categories mentioned above.

The following options are available:

-t Report the number of characters of terminal IO per

second as well.

-i Report the percentage of time spend in each of the four categories mentioned above, the percentage of time each disk controller was active (seeking or transferring, the percentage of time any disk drive was active, and the percentage of time spent in 'IO wait:' idle, but with a disk drive active.

-s Report the raw timing information for each active disk drive. The information consists of; the disk controller name and drive number, the disk's transfer rate (microseconds/word), the percentage of the total system time that the drive had I/O activity, the number of transfers on that drive, and the number of words transferred by the drive.

-b Report on the usage of I/O buffers. The report gives; the number of buffers in the pool, the number of buffered reads, number of read-ahead blocks, number of buffer cache hits, number of buffered writes, and the number of I/O operations on each buffer starting with the first one.

-d Print the date and time at the head of the report.

-a Print the total time in minutes at the end of the report.

FILES

/dev/mem - system memory
/unix - namelist

RESTRICTIONS

The accuracy of the iostat reports is subject to the sixtieth of a second granularity of the system clock.

All disk I/O statistics produced by iostat are approximate, the RA disk statistics are more approximate than others.

NAME

ipatch - print/modify contents of an inode

SYNOPSIS

/etc/ipatch filesystem inode

DESCRIPTION

Ipatch is an interactive program which allows the super-user to print and/or modify the contents of an inode. Filesystem is the name of the special file for the file system containing the inode to be patched and inode is the number of the inode to be patched. Printing inodes may be done freely, but patching should be done only on dismounted file systems.

Once invoked, the ipatch program prints a '.' prompt, indicating that it is ready to execute one of the following commands:

q	Quit, exit from ipatch.
p	Print the contents of the inode.
w	Write the patched inode out to the file system.

The following ipatch commands require numbers as arguments. If the first digit of a number is zero it is interpreted as octal, otherwise as decimal.

f #	Change the inode flags to #.
l #	Change the link count to #.
u #	Change the user ID to #.
g #	Change the group ID to #.
s #	Change the file size to # bytes.
a #addr #	Change the contents of the element of the inode's address array, specified by #addr, to #.

FILES

There is no default file system name.

SEE ALSO

clri(1), ncheck(1), icheck(1), fsck(1)

DIAGNOSTICS

Ipatch prints '?' if it cannot execute a command.

RESTRICTIONS

Can be very dangerous if not properly used.

NAME

join - relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of file1 and file2. If file1 is '-', the standard input is used.

File1 and file2 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 file1 and file2 that have identical join fields. The output line normally consists of the common field, then the rest of the line from file1, then the rest of the line from file2.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

-an In addition to the normal output, produce a line for each unpairable line in file n, where n is 1 or 2.

-e s Replace empty output fields by string s.

-jn m
Join on the mth field of file n. If n is missing, use the mth field in each file.

-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.

-tc Use character c as a separator (tab character). Every appearance of c in a line is significant.

SEE ALSO

sort(1), comm(1), awk(1)

RESTRICTIONS

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(1) are wildly incongruous.

NAME

kill - terminate a process with extreme prejudice

SYNOPSIS

kill [-signo] processid ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. If a signal number preceded by '-' is given as first argument, that signal is sent instead of terminate (see signal(2)). This will kill processes that do not catch the signal; in particular 'kill -9 ...' is a sure kill.

By convention, if process number 0 is specified, all members in the process group (i.e. processes resulting from the current login) are signaled.

The killed processes must belong to the current user unless he is the superuser. To shut the system down to single user mode use the operator services program. Refer to Chapter 5 of the ULTRIX-11 System Management Guide for operator services program information.

The process number of an asynchronous process started with '&' is reported by the shell. Process numbers can also be found by using ps(1).

SEE ALSO

ps(1), kill(2), signal(2)

NAME

ld - loader

SYNOPSIS

ld [option] file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object files are given, and ld combines them, producing an object module which can be either executed or become the input for a further ld run. (In the latter case, the -r option must be given to preserve the relocation bits.) The output of ld is left on a.out. This file is made executable only if no errors occurred during the load.

The overlay loader is used for the generation of the overlay text unix kernel and user overlay programs.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by ranlib(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named '__.SYMDEF', then it is understood to be a dictionary for the library such as produced by ranlib; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for -l, they should appear before the file names.

-s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by strip(1).

-u Take the following argument as a symbol and enter it as

LD(1)

undefined in the symbol table. 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.

-lx This option is an abbreviation for the library name '/lib/libx.a', where x is a string. If that does not exist, ld tries '/usr/lib/libx.a'. A library is searched when its name is encountered, so the placement of a -l is significant.

-x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

-X Save local symbols except for those whose names begin with 'L'. This option is used by cc(1) to discard internally generated labels while retaining symbols local to routines.

-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.

-d Force definition of common storage even if the -r flag is present.

-n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text.

-i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and -n is that here the data starts at location 0.

-o The name argument after -o is used as the name of the ld output file, instead of a.out.

-e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

-O This is an overlay file. This option will cause the entire text segment of a running process to be overlaid , retaining the current data segment, when exec(2) is called. Shared data must have the same layout as in the program overlaid.

-D The next argument is a decimal number that sets the size of the data segment.

-Z Marks the beginning of an overlay text segment. The object modules listed up to the next -Z or -L option are placed in the next overlay (numerically).

-L Marks the end of all overlays. Any further routines or libraries go into the base segment.

FILES

/lib/lib*.a	libraries
/usr/lib/lib*.a	more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), f77(1), ranlib(1)

RESTRICTIONS

The -O option is untested and probably does not work.

LEARN(1)

NAME

learn - computer aided instruction about UNIX

SYNOPSIS

learn [-directory] [subject [lesson [speed]]]

DESCRIPTION

Learn gives CAI courses and practice in the use of UNIX. To get started simply type 'learn'. The program will ask questions to find out what you want to do. The questions may be bypassed by naming a subject, and the last lesson number that learn told you in the previous session. You may also include a speed number that was given with the lesson number (but without the parentheses that learn places around the speed number). If lesson is '-', learn prompts for each lesson; this is useful for debugging.

The subjects presently handled are

editor
eqn
files
macros
morefiles
C

The special command 'bye' terminates a learn session.

The -directory option allows one to exercise a script in a nonstandard place.

FILES

/usr/lib/learn and all dependent directories and files

RESTRICTIONS

The main strength of learn, that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Such lessons may be skipped, but it takes some sophistication to recognize the situation.

NAME

lex - generator of lexical analysis programs

SYNOPSIS

lex [-tvfn] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text. The input files (standard input default) contain regular expressions to be searched 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 thus:

```
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 following lex program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A-Z] putchar(yytext[0]+'a'-'A');
[ ]+$
[ ]+  putchar(' ');
```

The options have the following meanings.

-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.

-n Opposite of -v; -n is default.

-f 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

SEE ALSO

yacc(1)

M. E. Lesk and E. Schmidt, LEX - Lexical Analyzer Generator

LINT(1)

NAME

lint - a C program verifier

SYNOPSIS

lint [-abchnpuvx] file ...

DESCRIPTION

Lint attempts to detect features of the C program files which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the files are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to lint; these libraries are referred to by a conventional name, such as '-lm', in the style of ld(1).

Any number of the options in the following list may be used. The -D, -U, and -I options of cc(1) are also recognized as separate arguments.

p Attempt to check portability to the IBM and GCOS dialects of C.

h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

b Report break statements that cannot be reached. (This is not the default because, unfortunately, most lex and many yacc outputs produce dozens of such comments.)

v Suppress complaints about unused arguments in functions.

x Report variables referred to by extern declarations, but never used.

a Report assignments of long values to int variables.

c Complain about casts which have questionable portability.

u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable

for running lint on a subset of files out of a larger program).

n Do not check compatibility against the standard library.

Exit(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of lint:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*VARARGSn*/
suppresses 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.

/*NOSTRICT*/
shuts off strict type checking in the next expression.

/*ARGSUSED*/
turns on the -v option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

FILES

/usr/lib/lint[12] programs
/usr/lib/llib-lc declarations for standard functions
/usr/lib/llib-port declarations for portable functions

SEE ALSO

cc(1)
S. C. Johnson, Lint, a C Program Checker

LN(1)

NAME

ln - make a link

SYNOPSIS

ln [-f] name1 [name2]

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There is no way to distinguish a link to a file from its original directory entry; any changes in the file are effective independently of the name by which the file is known.

The -f option allows the super-user to link to a directory.

Ln creates a link to an existing file name1. If name2 is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of name1.

It is forbidden to link across file systems.

SEE ALSO

rm(1)

NAME

login - sign on

SYNOPSIS

login [username]

DESCRIPTION

The login command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See 'How to Get Started' for how to dial up initially.

If login is invoked without an argument, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of .mail and message-of-the-day files. Login initializes the user and group IDs and the working directory, and sets the HOME, PATH, TERM, SHELL and USER environment variables. It then executes a command interpreter (usually sh(1) or csh(1)) according to specifications found in a password file. If the command interpreter is /bin/csh, the new line discipline is entered, otherwise the terminal stop characters are set to be undefined (see stty(1)). Argument 0 of the command interpreter starts with a '-'.

Login is recognized by sh(1) and csh(1) and executed directly (without forking).

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file

SEE ALSO

init(8), newgrp(1), getty(8), mail(1), passwd(1), passwd(5)

DIAGNOSTICS

'Login incorrect,' if the name or the password is bad.
'No Shell', 'cannot open password file', 'no directory': consult a programming counselor.

NAME

logins - enable user logins
nologins - disable user logins

SYNOPSIS

logins
nologins

DESCRIPTION

The logins and nologins commands allow the super-user to selectively enable and disable user logins. The super-user may login as 'root' on any terminal even when logins are disabled.

FILES

/etc/loglock disables user logins
/etc/sdloglock system shutdown in progress

SEE ALSO

login(1), shutdown(8)

DIAGNOSTICS

'No Logins', a user attempted to login while logins were disabled.

'LOGINS DISABLED', printed when the super-user logs in and user logins are disabled.

NAME

look - find lines in a sorted list

SYNOPSIS

look [-df] string [file]

DESCRIPTION

Look consults a sorted file and prints all lines that begin with string. It uses binary search.

The options d and f affect comparisons as in sort(1):

d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

f Fold. Upper case letters compare equal to lower case.

If no file is specified, /usr/dict/words is assumed with collating sequence -df.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

lorder - find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see ar(1)) files. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by tsort(1) to find an ordering of a library suitable for one-pass access by ld(1).

This brash one-liner intends to build a new library from existing '.o' files.

```
ar cr library 'lorder *.o | tsort'
```

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld(1), ar(1)

RESTRICTIONS

The names of object files, in and out of libraries, must end with '.o'; nonsense results otherwise.

NAME

lpq - show lineprinter queue

SYNOPSIS

lpq [option] [job_numbers ...] [user_names ...]

DESCRIPTION

Lpq displays the files that are queued for printing on a line printer. Any job numbers or user names entered with the command will limit the output to jobs with the appropriate job numbers or that have been printed by the users whose names were given. The following options are available:

-l Display the queue with a long format.

-Pprinter Display the queue for printer printer.

FILES

/usr/spool/lpd/lock
/usr/spool/lpd/cf* data file
/usr/spool/lpd/df* daemon control file
/usr/spool/lpd/tf* temporary version of control file

SEE ALSO

lpq(1), lprm(1), printcap(5), lpd(8), ulf(8)

LPR(1)

NAME

lpr - line printer spooler

SYNOPSIS

lpr [option] ... [file] ...

DESCRIPTION

Lpr causes the files to be queued for printing on a line printer. If no files are named, the standard input is read. The following options are available:

-r	Remove the file when it has been queued.
-c	Copy the file to insulate against changes that may happen before printing.
-m	Report by mail(1) when printing is complete.
-Jname	Print name instead of file name on header page.
-p	Pipe each file through pr before printing it.
-hname	Use name instead of file name on pr's header.
-Cname	Use name instead of hostname on header.
-inn	Indent output nn spaces (8 if just -i).
-#nn	Print nn copies of each file.
-Pprinter	Route output to alternate printer printer.

FILES

/usr/spool/lpd/lock	
/usr/spool/lpd/cf*	data file
/usr/spool/lpd/df*	daemon control file
/usr/spool/lpd/tf*	temporary version of control file
/usr/adm/lpacct	accounting records

SEE ALSO

lpq(1), lprm(1), printcap(5), lpd(8), ulf(8)

NAME

lprm - delete jobs from the lineprinter queue

SYNOPSIS

lprm [option] [job_numbers ...] [user_names ...]

DESCRIPTION

Lprm deletes the files from the print queue. Lprm will attempt to delete members whose job numbers or user names have been entered with the command. No one except the super-user may remove another user's files. The following options are available:

- Delete all jobs owned by the issuer of the command.

-Pprinter Delete jobs from the queue of the alternate printer printer.

FILES

/usr/spool/lpd/lock
/usr/spool/lpd/cf* data file
/usr/spool/lpd/df* daemon control file
/usr/spool/lpd/tf* temporary version of control file

SEE ALSO

lpq(1), lprm(1), printcap(5), lpd(8), ulf(8)

NAME

lpset - set line printer parameters

SYNOPSIS

/etc/lpset [-r]

/etc/lpset flag ind line col

DESCRIPTION

The lpset command with no arguments displays the current printer parameter values. Only the super-user may execute lpset. The printer must be quiescent when lpset is executed.

The -r option resets the parameters to the default values (flag = 0, ind = 0, line = 66, col = 132).

The parameters may be changed by specifying the new value of each parameter to lpset , as follows:

flag — Flag bits are set by oring their value with the flag argument.

The FFCLOSE (010) flag causes a page eject on LP close. This flag is normally reset, because the spooler lpr(1) ejects a page after each file is printed.

The CAP (020) should be set for 64-character set printers, uppercase only.

ind — Number of character spaces to indent from the left margin.

line — Number of lines to print per page.

col — The column width, i.e., number of characters per line.

Parameter changes may be made automatically by including the lpset command in the /etc/rc file.

FILES

/unix - system namelist
/dev/lp - printer special file

SEE ALSO

lpr(1), lp(4), lpd(8), init(8)

DIAGNOSTICS

Error messages will be printed if a parameter value is out of range, the printer file (/dev/lp) cannot be opened, or the printer is active.

NAME

ls - list contents of directory

SYNOPSIS

ls [-abcdfgilmqrstuxlCFR] name ...

DESCRIPTION

For each directory argument, ls lists the contents of the directory; for each file argument, ls repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The format chosen depends on whether the output is going to a teletype, and may also be controlled by option flags. The default format for a teletype is to list the contents of directories in multi-column format, with the entries sorted down the columns. (Files which are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. This is because the individual file names may be arbitrarily long.) If the standard output is not a teletype, the default format is to list one entry per line. Finally, there is a stream output format in which files are listed across the page, separated by ',' characters. The -m flag enables this format; when invoked as l this format is also used.

There are an unbelievable number of options:

-l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers.

-t Sort by time modified (latest first) instead of by name, as is normal.

-a List all entries; usually '.' and '..' are suppressed.

-s Give size in blocks, including indirect blocks, for each entry.

-d If argument is a directory, list only its name, not its contents (mostly used with -l to get status on directory).

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

LS(1)

-u Use time of last access instead of last modification for sorting (-t) or printing (-l).

-c Use time of file creation for sorting or printing.

-i Print i-number in first column of the report for each file listed.

-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.

-g Give group ID instead of owner ID in long listing.

-m force stream output format

-1 force one entry per line output format, e.g. to a teletype

-C force multi-column output, e.g. to a file or a pipe

-q force printing of non-graphic characters in file names as the character '?'; this normally happens only if the output device is a teletype

-b force printing of non-graphic characters to be in the \ddd notation, in octal.

-x force columnar printing to be sorted across rather than down the page; this is the default if the last character of the name the program is invoked with is an 'x'.

-F cause directories to be marked with a trailing '/' and executable files to be marked with a trailing '*'; this is the default if the last character of the name the program is invoked with is a 'f'.

-R recursively list subdirectories encountered.

The mode printed under the -l option contains 11 characters which are interpreted as follows: the first character is

d if the entry is a directory;
b if the entry is a block-type special file;
c if the entry is a character-type special file;
m if the entry is a multiplexor-type character special file;
- if the 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 to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
- if the indicated permission is not granted.

The group-execute permission character is given as s if the file has set-group-ID mode; likewise the user-execute permission character is given as s if the file has set-user-ID mode.

The last character of the mode (normally 'x' or '-') is t if the 1000 bit of the mode is on. See chmod(1) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd to get user ID's for 'ls -l'.
/etc/group to get group ID's for 'ls -g'.

RESTRICTIONS

Newline and tab are considered printing characters in file names.

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.

Column widths choices are poor for terminals which can tab.

M11(1)

NAME

m11 - Macro-11 assembler for UNIX

SYNOPSIS

m11 [option1 option2 ...] file1 file2 ... filen

DESCRIPTION

M11 assembles the concatenation of the specified files (file1, etc.) and terminates when an '.end' statement is encountered. The resulting object file is usually named filen.obj (see below). If a file argument, filei does not contain a '.' in its name, the file filei.m11 will be sought before filei itself.

Options, if desired, may appear anywhere in the command, and are chosen from the following list. All options are interpreted before any files are read.

-ls Produce an assembly listing and place in filen.lst

-lt Produce an assembly listing on the standard output.

-fl If coupled with the -ls or -lt directives, makes the listing have a shortened format. It is shorthand for -nl:seq:loc:bin:bex:me:meb:ttm:toc:sym.

-uc Simulate an initial .dsabl lc directive. Force all characters in macro definitions to be upper case. This flag makes lower-case handling in m11 compatible with the DEC Macro-11 assemblers.

-um Force all characters in macro definitions to be upper case. This flag makes lower-case handling in this release of m11 compatible with previous versions of m11.

-de Make all option choices needed to make assembly mimic DEC Macro-11. Implies (inter alia) the -uc flag. This includes the Johns Hopkins asm assembler.

-ha Make all option choices needed to make assembly mimic earlier (Harvard) releases of m11. This implies the -um flag. Default .psect and .csect attributes are set up in the funny Harvard way.

-mx Produce a listing of the source program as it appears after macro expansion. Macro calls, conditional directives and so on appear in the listing as comments. Listing appears on standard output. No machine code is generated or listed. This option is meant to

correspond to the -E or -P options of the C compiler cc(1).

-my Like -mx, except that macro calls and conditional directives do not show up in the listing.

-10 Generate an error whenever op codes not in the PDP-11 'standard instruction set' are encountered. Programmers writing for a PDP 11/10 can catch instructions illegal for that machine by using this argument.

-dp:args
The default attributes for a .psect or unnamed .csect are redefined, using the colon-separated list args of valid .psect attributes.

-da:args
The default attributes for an .asect are redefined.

-dp:c
The default attributes for a named .csect are redefined.

-li:arglist
Simulate an initial .list arglist directive. All .list and .nlist directives in the program text which attempt to change the settings established with the -li flag will be ignored.

-nl:arglist
Like -li:arglist, but for the .nlist directive.

-en:arglist
Similarly, for the .enabl directive.

-ds:arglist
Similarly, but for the .dsabl directive.

-cr:arglist
Produces a cross-reference listing. If the -ls option is also included, the cross-reference listing will follow the assembly listing in filen.lst. References which are tagged with the symbol # are definitions. References tagged with * are destructive references: the value of the symbol or variable in question is changed. Arglist consists of colon-separated keywords from the following set. The keywords may be prefix abbreviated:

sym All user-defined symbols are indexed.

mac All macro names are indexed.

per All uses of permanent symbols - op codes, directives, etc - are indexed.

pse All psect names are indexed. For compatibility with the RT-11 CREF program, the argument cse is synonymous with pse.

err All errors are indexed.

reg All register uses are indexed.

If no arglist is specified the default sym:mac:err is used. In the listing page and line numbers for uses of symbols are followed by a # sign if the symbol is defined and by a * sign if the symbol is modified.

-lp Same as -ls, but also spools filen.lst for printing upon completion.

-no No object file is produced. This is useful for syntax checking or list producing.

-xs:n
Allots nK words of extra space for symbol table and macro storage. NOTE: This option is currently inoperative: m11 automatically allots core for its tables as needed.

-xx Debug flag: generate all kinds of weird hack flack.

-ns No symbol table is included in the object file (thus ddt knows of no symbols from this assembly).

-sx Make the symbol table contain 'local symbols' as well as ordinary symbols.

-u Treat form feed characters as spaces. This will make m11's idea of line numbers coincide with the UNIX text editors. Macro-11 statements containing imbedded form feed characters will be parsed differently when the -u flag is in effect.

-na:file
Override the convention of using last name as file name. Instead, use names file.obj and file.lst for object and listing files.

NOTES

This implementation of Macro-11 is a distant hand-me down from an old DEC Macro-11 modified at Harvard University in the early 1970's. It is grubby with smudges by Brent Byer, F. J. Howard, Bob Bowering, and Jim Reeds. It does not

implement keyword arguments such as are described in section 7.3.6 of the DEC manual. The .enabl abs option does not work. Listing control is by default .list ttm. Unlike earlier editions of m11 at UCB and at Harvard, it does treat immediate constants of floating point operations correctly: see the last paragraph of section 6.4.2 on the middle of page 6-27 of the DEC manual. M11 has several directives not described in the DEC manual. See the New UCB M11 Manual. The default attributes for .psects are different from those described in the DEC manual, but may be changed by the -dp flag. The .enabl glb feature is implemented: undefined symbols are taken as undefined global externals.

FILES

/usr/lib/sysmac.sml	system macro library (for .mcall directive)
filen.xrf	intermediate cross-reference temporary file
lpr(1)	spooler
/usr/ucb/macxrf	cross-reference post-processor

SEE ALSO

PDP-11 MACRO-11 Language Reference Manual , Digital Equipment Corp. Order No. AA-5075A-TC, August 1977.
New UCB M11 Manual , notional document by Jim Reeds.

M4(1)

NAME

m4 - macro processor

SYNOPSIS

m4 [files]

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 arguments, or if an argument is '-', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

name(arg1,arg2, . . . , argn)

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

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, its arguments are collected by searching for a matching right parenthesis. 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.

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 — removes the definition of the macro named in its 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 on UNIX versions of m4.

changequote
Change quote characters to the first and second arguments. Changequote without arguments restores the original values (i.e., `').

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 its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert causes 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 returns the value of the current output stream.

dnl reads and discards 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 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation); relationals; parentheses.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr returns a substring of its first argument. The

M4(1)

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 transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is identical to include, except that it says nothing if the file is inaccessible.

syscmd executes the UNIX command given in the first argument. No value is returned.

maketemp fills in a string of XXXXX in its argument with the current process id.

errprint prints its argument on the diagnostic output file.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.

SEE ALSO

B. W. Kernighan and D. M. Ritchie, The M4 Macro Processor

NAME

mail - send and receive mail

SYNTAX

```
mail [ -v ] [ -i ] [ -n ] [ -s subject ] [ user ... ]
mail [ -v ] [ -i ] [ -n ] -f [ name ]
mail [ -v ] [ -i ] [ -n ] -u user
```

INTRODUCTION

Mail is a intelligent mail processing system, which has a command syntax reminiscent of ed with lines replaced by messages.

The -v flag puts mail into verbose mode; the details of delivery are displayed on the users terminal. The -i flag causes tty interrupt signals to be ignored. This is particularly useful when using mail on noisy phone lines. The -n flag inhibits the reading of /usr/lib/Mail.rc.

Sending mail. To send a message to one or more other people, mail can be invoked with arguments which are the names of people to send to. You are then expected to type in your message, followed by an EOT (control-D) at the beginning of a line. A subject may be specified on the command line by using the -s flag. (Only the first argument after the -s flag is used as a subject; be careful to quote subjects containing spaces.) The section below, labeled Replying to or originating mail, describes some features of mail available to help you compose your letter.

Reading mail. In normal usage mail is given no arguments and checks your mail out of the post office, then prints out a one line header of each message there. The current message is initially the first message (numbered 1) and can be printed using the print command (which can be abbreviated p). You can move among the messages much as you move between lines in ed, either with the commands `+' and `-' moving backwards and forwards or with simple numbers.

Disposing of mail. After examining a message you can delete (d) the message or reply (r) to it. Deletion causes the mail program to forget about the message. This is not irreversible; the message can be undeleted (u) by giving its number, or the mail session can be aborted by giving the exit (x) command. Deleted messages will, however, usually disappear never to be seen again.

Specifying messages. Commands such as print and delete can be given a list of message numbers as arguments to apply to a number of messages at once. Thus ``delete 1 2'' deletes messages 1 and 2, while ``delete 1-5'' deletes messages 1 through 5. The special name ``*'' addresses all messages,

and ``$'' addresses the last message; thus the command top which prints the first few lines of a message could be used in ``top *'' to print the first few lines of all messages.

Replying to or originating mail. You can use the reply command to set up a response to a message, sending it back to the person who it was from. Text you then type in, up to an end-of-file, defines the contents of the message. While you are composing a message, mail treats lines beginning with the character `~' specially. For instance, typing ``~m'' (alone on a line) will place a copy of the current message into the response right shifting it by a tabstop. Other escapes will set up subject fields, add and delete recipients to the message and allow you to escape to an editor to revise the message or to a shell to run some commands. (These options are given in the summary below.)

Ending a mail processing session. You can end a mail session with the quit (q) command. Messages which have been examined go to your mbox file unless they have been deleted in which case they are discarded. Unexamined messages go back to the post office. The -f option causes mail to read in the contents of your mbox (or the specified file) for processing; when you quit, mail writes undeleted messages back to this file. The -u flag is a short way of doing "mail -f /usr/spool/mail/user".

Personal and systemwide distribution lists. It is also possible to create a personal distribution lists so that, for instance, you can send mail to ``cohorts'' and have it go to a group of people. Such lists can be defined by placing a line like

alias cohorts bill ozalp jkf mark kridle@ucbcory

in the file .mailrc in your home directory. The current list of such aliases can be displayed with the alias (a) command in mail. System wide distribution lists can be created by editing /usr/lib/aliases, see aliases(5) and sendmail(8); these are kept in a different syntax. In mail you send, personal aliases will be expanded in mail sent to others so that they will be able to reply to the recipients. System wide aliases are not expanded when the mail is sent, but any reply returned to the machine will have the system wide alias expanded as all mail goes through sendmail.

Network mail (ARPA, UUCP, Berknet) See mailaddr(7) for a description of network addresses.

Mail has a number of options which can be set in the .mailrc file to alter its behavior; thus ``set askcc'' enables the ``askcc'' feature. (These options are summarized below.)

SUMMARY

(Adapted from the `Mail Reference Manual')

Each command is typed on a line by itself, and may take arguments following the command word. The command need not be typed in its entirety - the first command which matches the typed prefix is used. For commands which take message lists as arguments, if no message list is given, then the next message forward which satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, mail types ``No applicable messages'' and aborts the command.

- Goes to the previous message and prints it out. If given a numeric argument n, goes to the n-th previous message and prints it.

? Prints a brief summary of commands.

! Executes the UNIX shell command which follows.

Print (P) Like print but also prints out ignored header fields. See also print and ignore.

Reply (R) Reply to originator. Does not reply to other recipients of the original message.

Type (T) Identical to the Print command.

alias (a) With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, creates an new or changes an on old alias. These aliases are in effect for the current mail session only.

alternates (alt) The alternates command is useful if you have accounts on several machines. It can be used to inform mail that the listed addresses are really you. When you reply to messages, mail will not send a copy of the message to any of the addresses listed on the alternates list. If the alternates command is given with no argument, the current set of alternate names is displayed.

chdir (ch) Changes the user's working directory to that specified. If no directory is given, then changes to the user's login directory.

copy (co) The copy command does the same thing that

MAIL(1)

save does, except that it does not mark the messages it is used on for deletion when you quit.

delete — (d) Takes a list of messages as argument and marks them all as deleted. Deleted messages will not be saved in mbox, nor will they be available for most other commands.

dp — (also dt) Deletes the current message and prints the next message. If there is no next message, mail says ``at EOF.''

edit — (e) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.

exit — (ex or x) Effects an immediate return to the Shell without modifying the user's system mailbox, his mbox file, or his edit file in -f.

file — (fi) The same as folder.

folders — List the names of the folders in your folder directory.

folder — (fo) The folder command switches to a new mail file or folder. With no arguments, it tells you which file you are currently reading. If you give it an argument, it will write out changes (such as deletions) you have made in the current file and read in the new file. Some special conventions are recognized for the name. # means the previous file, % means your system mailbox, %user means user's system mailbox, & means your ~/mbox file, and +folder means a file in your folder directory.

from — (f) Takes a list of messages and prints their message headers in the order that they appear in the mail directory, not in the order given in the list.

headers — (h) Lists the current range of headers, which is an 18 message group. If a ``+'' argument is given, then the next 18 message group is printed, and if a ``-'' argument is given, the previous 18 message group is printed.

help — A synonym for ?

hold — (ho, also preserve) Takes a message list and marks each message therein to be saved in the

user's system mailbox instead of in mbox. Does not override the delete command.

ignore — Add the list of header fields named to the ignored list. Header fields in the ignore list are not printed on your terminal when you print a message. This command is very handy for suppression of certain machine-generated header fields. The Type and Print commands can be used to print a message in its entirety, including ignored fields. If ignore is executed with no arguments, it lists the current set of ignored fields.

mail — (m) Takes as argument login names and distribution group names and sends mail to those people.

mbox — Indicate that a list of messages be sent to mbox in your home directory when you quit. This is the default action for messages if you do not have the hold option set.

next — (n like + or CR) Goes to the next message in sequence and types it. With an argument list, types the next matching message.

preserve — (pre) A synonym for hold.

print — (p) Takes a message list and types out each message on the user's terminal.

quit — (q) Terminates the session, saving all undeleted, unsaved messages in the user's mbox file in his login directory, preserving all messages marked with hold or preserve or never referenced in his system mailbox, and removing all other messages from his system mailbox. If new mail has arrived during the session, the message ``You have new mail'' is given. If given while editing a mailbox file with the -f flag, then the edit file is rewritten. A return to the Shell is effected, unless the rewrite of edit file fails, in which case the user can escape with the exit command.

reply — (r) Takes a message list and sends mail to the sender and all recipients of the specified message. The default message must not be deleted.

respond — A synonym for reply.

save — (s) Takes a message list and a filename and

appends each message to the end of the file. The messages are saved in the order in which they appear in the mail directory, not in that given in the message list. The filename in quotes, followed by the line count and character count is echoed on the user's terminal.

set — (se) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form ``option=value'' or ``option.''

shell — (sh) Invokes an interactive version of the shell.

size — Takes a message list and prints out the size (in characters) of each message. The size of the messages are printed in the order that they appear in the mail directory, not in that given in the list.

source — (so) The source command reads mail commands from a file.

top — Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable toplines and defaults to five.

type — (t) A synonym for print.

unalias — Takes a list of names defined by alias commands and discards the remembered groups of users. The group names no longer have any significance.

undelete — (u) Takes a message list and marks each one as not being deleted.

unset — Takes a list of option names and discards their remembered values; the inverse of set.

visual — (v) Takes a message list and invokes the display editor on each message.

write — (w) A synonym for save.

xit — (x) A synonym for exit.

z — Mail presents message headers in windowfuls as described under the headers command. You can move mail's attention forward to the next window with the z command. Also, you can move to the previous window by using z-.

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name ``tilde escape'' is somewhat of a misnomer since the actual escape character can be set by the option escape.

~!command Execute the indicated shell command, then return to the message.

~c name ... Add the given names to the list of carbon copy recipients.

~d Read the file ``dead.letter'' from your home directory into the message.

~e Invoke the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~f messages Read the named messages into the message being sent. If no messages are specified, read in the current message.

~h Edit the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field by using the current terminal erase and kill characters.

~m messages Read the named messages into the message being sent, shifted right one tab. If no messages are specified, read the current message.

~p Print out the message collected so far, prefaced by the message header fields.

~q Abort the message being sent, copying the message to ``dead.letter'' in your home directory if save is set.

~r filename Read the named file into the message.

~s string Cause the named string to become the current subject field.

~t name ... Add the given names to the direct recipient list.

~v Invoke an alternate editor (defined by the VISUAL option) on the message collected so far. Usually, the alternate editor will be a screen editor. After you quit the editor, you may resume appending text to the end of your

MAIL(1)

message.

~w filename Write the message onto the named file.

~|command Pipe the message through the command as a filter. If the command gives no output or terminates abnormally, retain the original text of the message. The command fmt(1) is often used as command to rejustify the message.

~~string Insert the string of text in the message prefaced by a single ~. If you have changed the escape character, then you should double that character in order to send it.

Options are controlled via the set and unset commands. Options may be either binary, in which case it is only significant to see whether they are set or not, or string, in which case the actual value is of interest. The binary options include the following:

append Causes messages saved in mbox to be appended to the end rather than prepended. (This is set in /usr/lib/Mail.rc on version 7 systems.)

ask Causes mail to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field will be sent.

askcc Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

autoprint Causes the delete command to behave like dp - thus, after deleting a message, the next one will be typed automatically.

debug Setting the binary option debug is the same as specifying -d on the command line and causes mail to output all sorts of information useful for debugging mail.

dot The binary option dot causes mail to interpret a period alone on a line as the terminator of a message you are sending.

hold This option is used to hold messages in the system mailbox by default.

ignore — Causes interrupt signals from your terminal to be ignored and echoed as @'s.

ignoreeof — An option related to dot is ignoreeof which makes mail refuse to accept a control-d as the end of a message. Ignoreeof also applies to mail command mode.

msgprompt — When sending mail, prompts you for the message text and indicates how to terminate the message.

metoo — Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.

nosave — Normally, when you abort a message with two RUBOUT, mail A copies the partial letter to the file dead.letter'' in your home directory. Setting the binary option nosave prevents this.

quiet — Suppresses the printing of the version when first invoked.

verbose — Setting the option verbose is the same as using the -v flag on the command line. When mail runs in verbose mode, the actual delivery of messages is displayed on he users terminal.

The following options have string values:

EDITOR — Pathname of the text editor to use in the edit command and ~e escape. If not defined, then a default editor is used.

SHELL — Pathname of the shell to use in the ! command and the ~! escape. A default shell is used if this option is not defined.

VISUAL — Pathname of the text editor to use in the visual command and ~v escape.

crt — The valued option crt is used as a threshold to determine how long a message must be before more is used to read it.

escape — If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.

MAIL(1)

folder — The name of the directory to use for storing folders of messages. If this name begins with a `/', mail considers it to be an absolute pathname; otherwise, the folder directory is found relative to your home directory.

record — If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not so saved.

toplines — If defined, gives the number of lines of a message to be printed out with the top command; normally, the first five lines are printed.

FILES

/usr/spool/mail/*	post office
~/mbox	your old mail
~/.mailrc	file giving initial mail commands
/tmp/R#	temporary for editor escape
/usr/lib/Mail.help*	help files
/usr/lib/Mail.rc	system initialization file
Message*	temporary for editing messages

SEE ALSO

binmail(1), from(1)
`The Mail Reference Manual'

NAME

mail - send or receive mail among users

SYNOPSIS

/bin/mail person ...
mail [-r] [-q] [-p] [-f file]

DESCRIPTION

Mail with no argument prints a user's mail, message-by-message, in last-in, first-out order; the optional argument -r causes first-in, first-out order. If the -p flag is given, the mail is printed with no questions asked; otherwise, for each message, mail reads a line from the standard input to direct disposition of the message.

newline
Go on to next message.

d Delete message and go on to the next.

p Print message again.

- Go back to previous message.

s [file] ...
Save the message in the named files ('mbox' default).

w [file] ...
Save the message, without a header, in the named files ('mbox' default).

m [person] ...
Mail the message to the named persons (yourself is default).

EOT (control-D)
Put unexamined mail back in the mailbox and stop.

q Same as EOT.

x Exit, without changing the mailbox file.

!command
Escape to the Shell to do command.

? Print a command summary.

An interrupt stops the printing of the current letter. The optional argument -q causes mail to exit after interrupts without changing the mailbox.

MAIL(1)

When persons are named, mail takes the standard input up to an end-of-file (or a line with just '.') and adds it to each person's 'mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A person is usually a user name recognized by login(1). To denote a recipient on a remote system, prefix person by the system name and exclamation mark (see uucp(1)).

The -f option causes the named file, e.g. 'mbox', to be printed as if it were the mail file.

Each user owns his own mailbox, which is by default generally readable but not writable. The command does not delete an empty mailbox nor change its mode, so a user may make it unreadable if desired.

When a user logs in he is informed of the presence of mail.

FILES

```
/usr/spool/mail/*   mailboxes
/etc/passwd    to identify sender and locate persons
mbox       saved mail
/tmp/ma*  temp file
dead.letter    unmailable text
uux(1)
```

SEE ALSO

xsend(1), write(1), uucp(1)

RESTRICTIONS

There is a locking mechanism intended to prevent two senders from accessing the same mailbox, but it is not perfect and races are possible.

NAME

make - maintain program groups

SYNOPSIS

make [-f makefile] [option] ... file ...

DESCRIPTION

Make executes commands in makefile to update one or more target names. Name is typically a program. If no -f option is present, 'makefile' and 'Makefile' are tried in order. If makefile is '-', the standard input is taken. More than one -f option may appear

Make updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target.

Sharp and newline surround comments.

The following makefile says that 'pgm' depends on two files 'a.o' and 'b.o', and that they in turn depend on '.c' files and a common file 'incl'.

```
pgm: a.o b.o
     cc a.o b.o -lm -o pgm
a.o: incl a.c
     cc -c a.c
b.o: incl b.c
     cc -c b.c
```

Makefile entries of the form

```
string1 = string2
```

are macro definitions. Subsequent appearances of $(string1) are replaced by string2. If string1 is a single character, the parentheses are optional.

Make infers prerequisites for files for which makefile gives no construction commands. For example, a '.c' file may be inferred as prerequisite for a '.o' file and be compiled to produce the '.o' file. Thus the preceding example can be done more briefly:

MAKE(1)

```
pgm: a.o b.o
     cc a.o b.o -lm -o pgm
a.o b.o: incl
```

Prerequisites are inferred according to selected suffixes listed as the 'prerequisites' for the special name '.SUFFIXES'; multiple lists accumulate; an empty list clears what came before. Order is significant; the first possible name for which both a file and a rule as described in the next paragraph exist is inferred. The default list is

```
.SUFFIXES: .out .o .c .e .r .f .y .l .s
```

The rule to create a file with suffix s2 that depends on a similarly named file with suffix s1 is specified as an entry for the 'target' s1s2. In such an entry, the special macro $* stands for the target name with suffix deleted, $@ for the full target name, $< for the complete list of prerequisites, and $? for the list of prerequisites that are out of date. For example, a rule for making optimized '.o' files from '.c' files is

```
.c.o: ; cc -c -O -o $@ $*.c
```

Certain macros are used by the default inference rules to communicate optional arguments to any resulting compilations. In particular, 'CFLAGS' is used for cc and f77(1) options, 'LFLAGS' and 'YFLAGS' for lex and yacc(1) options.

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target '.SILENT' is in makefile, or the first character of the command is '@'.

Commands returning nonzero status (see intro(1)) cause make to terminate unless the special target '.IGNORE' is in makefile or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target depends on the special name '.PRECIOUS'.

Other options:

-i Equivalent to the special entry '.IGNORE:'.

-k When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

-n Trace and print, but do not execute the commands needed to update the targets.

-t Touch, i.e. update the modified date of targets, without executing any commands.

-r Equivalent to an initial special entry '.SUFFIXES:' with no list.

-s Equivalent to the special entry '.SILENT:'.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1)
S. I. Feldman Make - A Program for Maintaining Computer Programs

RESTRICTIONS

Some commands return nonzero status inappropriately. Use -i to overcome the difficulty.
Commands that are directly executed by the shell, notably cd(1), are ineffectual across newlines in make.

MAN(1)

NAME

man - print sections of this manual

SYNOPSIS

man [option ...] [chapter] title ...

DESCRIPTION

Man locates and prints the section of this manual named title in the specified chapter. (In this context, the word 'page' is often used as a synonym for 'section'.) The title is entered in lower case. The chapter number does not need a letter suffix. If no chapter is specified, the whole manual is searched for title and all occurrences of it are printed.

Options and their meanings are:

-t Phototypeset the section using troff(1).

-n Print the section on the standard output using nroff(1).

-k Display the output on a Tektronix 4014 terminal using troff(1) and tc(1).

-e Appended or prefixed to any of the above causes the manual section to be preprocessed by neqn or eqn(1); -e alone means -te.

-w Print the path names of the manual sections, but do not print the sections themselves.

(default)
Copy an already formatted manual section to the terminal, or, if none is available, act as -n. It may be necessary to use a filter to adapt the output to the particular terminal's characteristics.

Further options, e.g. to specify the kind of terminal you have, are passed on to troff(1) or nroff. Options and chapter may be changed before each title.

For example:

man man

would reproduce this section, as well as any other sections named man that may exist in other chapters of the manual, e.g. man(7).

FILES

/usr/man/man?/*
/usr/man/cat?/*
/tmp/Man*

SEE ALSO

nroff(1), eqn(1), tc(1), man(7)

RESTRICTIONS

The manual is supposed to be reproducible either on a phototypesetter or on a terminal. However, on a terminal some information is necessarily lost.

MEMSTAT(1M)

NAME

memstat - print memory usage map

SYNOPSIS

memstat [ifn] [interval] [corefile] [namelist]

DESCRIPTION

The memstat program prints a map showing the utilization of all of memory. The optional argument 'i' specifies that the printout should be repeated every interval seconds. The 'f' option causes corefile to be used as memory instead of /dev/mem. The 'n' option declares that the system namelist should be obtained from namelist instead of /unix. Single or multiple options may be specified. The arguments interval, corefile, and namelist must occur in the same order as the options [ifn].

The following headings appear on the memstat printout:

Addr The octal address range of the section of memory being mapped.

Command The name of the entity occupying a section of memory. This may be the operating system, a process, a shared text segment, or free memory.

Pid The process ID number of the process named in command.

Size The total size in bytes of non-shared text processes or the size of the data segment of shared text processes.

Ublock The address and size of the U block (per process data) for the process.

Text Size and address of the processes text segment.

Data Size and address of the processes data segment.

Stack Size and address of the processes stack.

FILES

/unix - default system namelist
/dev/mem - default system memory

SEE ALSO

Refer to the ULTRIX-11 System Management Guide, Section 9.5.5 for an example of a memstat printout.

NAME

mesg - permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument n forbids messages via 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*
/dev

SEE ALSO

write(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

MKCONF(1M)

NAME

mkconf - generate configuration tables

SYNOPSIS

mkconf <?.cf

DESCRIPTION

The mkconf program is a key part of the internal workings of the system generation process. Mkconf is called by the system generation program (sysgen). The following documentation of the mkconf program is for informational purposes only, refer to the ULTRIX-11 System Management Guide, Chapter 2 a description of the sysgen program and its usage.

The mkconf program reads the (?.cf) system configuration file and creates the following files:

l.s	locore trap and interrupt vectors
c.c	configuration tables
mch0.s	machine language assist header file
dds.h	disk data structures header file
devmaj.h	major device number header file
ovload	overlay kernel load shell file

The configuration file contains device specifications , file system placement specifications, and system parameter values.

The device specifications use the following format:

#dev addr vector cn

where; # is the number of units, dev is the device mnemonic, addr is the device hardware address, vector is its interrupt vector address, and cn is the disk controller number for MSCP and MASSBUS disks. If addr and vector are omitted the standard device address and vector are used.

The following device mnemonics will be accepted by mkconf, not all are supported by the ULTRIX-11 system:

ct	(CAT - phototypesetter)
lp	(LP11)
dc	(DC11)
kl	(KL11/DL11-ABCD)
dl	(DL11-E)
dp	(DP11)
dn	(DN11)
dh	(DH11)
dhdm	(DM11-BB)
uh	(DHU11/DHV11)
du	(DU11)

```
dz    (DZ11)
dzv   (DZV11/DZQ11)
ra    (RA60/RA80/RA81)
rc    (RC25)
rq    (RD51/RD52/RX50)
hs    (RS03/RS04)
tk    (reserved)
rk    (RK03/RK05)
rl    (RL01/2)
tm    (TM11)
rp    (RP02/3)
hp    (RM02/3/5, RP04/5/6, ML11 on first RH)
hm    (RM02/3/5, RP04/5/6, ML11 on second RH)
hj    (RM02/3/5, RP04/5/6, ML11 on third RH)
hk    (RK06/7)
ht    (TM02/3)
ts    (TS11/TSV05/TK25/TU80)
hx    (RX02)
ul-4  user devices
```

The following file system placement and parameter specifications are also accepted:

root dev minor
The specified block device (e.g. hp) is used for the root. minor is a decimal number giving the minor device. This line must appear exactly once.

pipe dev minor
The specified block device is used to store pipes. If not given the root is used.

swap dev minor
The specified block device is used for swapping. If not given the root is used.

swplo number

nswap number
Sets the origin (block number) and size of the area used for swapping. By default, the not very useful numbers 4000 and 800.

eldev dev minor
The specified block device (e.g. hp) is used for the error log. minor is a decimal number giving the minor device.

elsb number

elnb number
Sets the origin (block number) and size of the area

NAME

mkdir - make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

rm(1)

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

MKCONF(1M)

used for the error log.

dump dev
Selects the crash dump device, i.e., disk, tape, or RX50.

dumplo number

dumphi number

dumpdn number
If the crash dump device is a disk, specifies the; starting block number, ending block number, and the disk unit number for the crash dump area, normally located in the swap area.

ov Causes mkconf to create the file "ovload" which is used by the makefile in /sys/conf to link the overlay ULTRIX-11 kernel. Must be specified if the overlay kernel is to be generated.

parameter number
Sets the specified parameter to number. Each parameter has a default value, which is used if the parameter is not specified. The valid parameters are; nbuf, ninode, nfile, nmount, maxuprc, ncall, nproc, ntext, nclist, canbsiz, hz, timezone, dstflag, ncargs, maxseg, msgbufs.

FILES

l.s, dds.h, devmaj.h, ovload, c.c, mch0.s - output files
?.cf - configuration input file

SEE ALSO

ULTRIX-11 System Management Guide, Chapter 2

MKFS(1M)

NAME

mkfs - construct a file system

SYNOPSIS

/etc/mkfs special proto/size [disk cpu] [fsname volname]
/etc/mkfs special proto/size [m n] [fsname volname]

DESCRIPTION

Mkfs constructs a file system by writing on the special file special according to the specification proto/size. The proto/size specification is either a prototype file containing instructions on how to build the file system, or a number specifying the size of the file system.

The prototype file contains tokens separated by spaces or new lines. The first token is the name of a file to be copied onto block zero as the bootstrap program, see boot(8). The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the i-list. The next set of tokens comprise the specification for the root file. 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(1).

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, mkfs 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 $.

If the proto/size specification consists of a string of digits, mkfs builds a file system with a single empty directory on it. The size of the file system is the value of proto/size interpreted as a decimal number. The number of i-nodes is calculated as a function of the file system size. The boot program is left uninitialized.

The optional arguments m and n are the file system interleave factors , used for optimal free list spacing. If m and n are not specified, the values m = 9 and n = 500 are used.

The preferred method of specifying optimal free list spacing is to allow mkfs to select the optimum values of m and n according to the type of disk and processor. The disk is the generic name of the disk (rl02, rp06, ra80, etc.) and cpu is the number of the processor (23, 34, 44, etc.).

The optional arguments fsname and volname are the file system and volume name labels to be recorded in the last 12 characters of the superblock (char fsname[6], volname[6];). If fsname and volname are omitted, these fields are left uninitialized. The labelit(1m) command may be use to initialize or change the fsname and volname labels after the file system has been created. Refer to volcopy(1m) for information on the labelit(1m) command.

To create a file system on an RL02 disk, unit zero, the following command would be used:

```
/etc/mkfs /dev/rrl07 20480 rl02 23 /user users
```

Refer to Appendix D of the ULTRIX-11 System Management Guide for disk partition size and layout information for all supported disks.

A sample prototype specification follows:

```
/usr/mdec/uboot
4872 55
d--777 3 1
usr  d--777 3 1
     sh   ---755 3 1 /bin/sh
     ken  d--755 6 1
          $
     b0   b--644 3 1 0 0
     c0   c--644 3 1 0 0
     $
$
```

MKFS(1M)

SEE ALSO

filsys(5), dir(5), boot(8), volcopy(1m),
ULTRIX-11 System Management Guide, Section 4.6

RESTRICTIONS

There should be some way to specify links.

NAME

mknod - build special file

SYNOPSIS

/etc/mknod name [c] [b] major minor

DESCRIPTION

Mknod makes a special file. The first argument is the name of the entry. The second 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 (e.g. unit, drive, or line number).

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file conf.c.

SEE ALSO

mknod(2), csf(1)

MKSTR(1)

NAME

mkstr - create an error message file by massaging C source

SYNTAX

mkstr [-] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified files, placing a massaged version of the input file in a file whose name consists of the specified prefix and the original name. A typical usage of mkstr would be:

```
mkstr pistrings xx *.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 xx.

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 new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly cat the error message file to see its contents. The massaged copy of the input file then contains an lseek pointer into the file which can be used to retrieve the message, i.e.:

```
char efilname[] =  "/usr/lib/pistrings";
int  efil = -1;

error(a1, a2, a3, a4)
{
     char buf[BUFSIZ];

     if (efil < 0) {
          efil = open(efilname, FATT_RDONLY);
          if (efil < 0) {
oops:
               perror(efilname);
               exit(1);
          }
```

```
            }
            if (lseek(efil, (long) a1, FSEEK_ABSOLUTE)
                || read(efil, buf, sizeof buf) <= 0)
                 goto oops;
            printf(buf, a2, a3, a4);
        }
```

The optional "-" causes the error messages to be placed at the end of the specified message file for recompiling part of a large program upon which mkstr has already been run.

SEE ALSO

xstr(1), lseek(2)

RESTRICTIONS

All the arguments except the name of the file to be processed are unnecessary.

AUTHORS

Bill Joy and Charles Haley

MORE(1)

NAME

more, page - file perusal filter for crt viewing

SYNOPSIS

more [-cdflsu] [-n] [+linenumber] [+/pattern] [name ...]

page more options

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

The command line options are:

-n An integer which is the size (in lines) of the window which more will use instead of the default.

-c More will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while more is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

-d More will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if more is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

-f This causes more to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if nroff output is being piped through ul, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen postions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus more may think that lines are longer than they actually are, and fold lines erroneously.

-l Do not treat ^L (form feed) specially. If this option is not given, more will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

-s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing nroff output, this option maximizes the useful information present on the screen.

-u Normally, more will handle underlining such as produced by nroff in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, more will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The -u option suppresses this processing.

+linenumber
Start up at linenumber.

+/pattern
Start up two lines before the line containing the regular expression pattern.

If the program is invoked as page, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and k - 1 rather than k - 2 lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the file /etc/termcap to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable MORE to pre-set any flags desired. For example, if you prefer to view files using the -c mode of operation, the csh command setenv MORE -c or the sh command sequence MORE='-c' ; export MORE would cause all invocations of more , including invocations by programs such as man and msgs , to use this mode. Normally, the user will place the command sequence which sets up the MORE environment variable in the .cshrc or .profile file.

If more is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when more pauses, and their effects, are as follows (i is an optional integer argument, defaulting to 1) :

i<space>
display i more lines, (or another screenful if no argument is given)

MORE(1)

^D display 11 more lines (a 'scroll'). If i is given, then the scroll size is set to i.

d same as ^D (control-D)

iz same as typing a space except that i, if present, becomes the new window size.

is skip i lines and print a screenful of lines

if skip i screenfuls and print a screenful of lines

q or Q
Exit from more.

= Display the current line number.

v Start up the editor vi at the current line.

h Help command; give a description of all the more commands.

i/expr
search for the i-th occurrence of the regular expression expr. If there are less than i occurrences of expr, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

in search for the i-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
invoke a shell with command. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the i-th next file given in the command line (skips to last file if n doesn't make sense)

i:p skip to the i-th previous file given in the command line. If this command is given in the middle of printing out a file, then more goes back to the beginning of the file. If i doesn't make sense, more skips back to the first file. If more is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q
exit from more (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). More will stop sending output, and will display the usual --More-- 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.

The terminal is set to noecho mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then more acts just like cat, except that a header is printed before each file (if there is more than one).

A sample usage of more in previewing nroff output would be

```
nroff -ms +2 doc.n | more -s
```

AUTHOR
Eric Shienbrood, minor revisions by John Foderaro and Geoffrey Peck

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

NAME

mount, umount - mount and dismount file system

SYNOPSIS

/etc/mount [special name [-r]]

/etc/umount special

/etc/mount -a

/etc/umount -a

DESCRIPTION

Mount announces to the system that a removable file system is present on the device special. The file name must exist already; it must be a directory (unless the root of the mounted file system is not a directory). It becomes the name of the newly mounted root. The optional last argument indicates that the file system is to be mounted read-only.

Umount announces to the system that the removable file system previously mounted on device special is to be removed.

These commands maintain a table of mounted devices. If invoked without an argument, mount prints the table. Only the superuser may mount and dismount file systems, any user may list the mount table.

The '-r' option declares that the file system is to be mounted read only. 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 '-a' option causes all file systems listed in the file system table (/etc/fstab) to be mounted or dismounted. To allow for nested file systems mounts, dismounts are done in reverse order, i.e., the last entry in the fstab is dismounted first.

FILES

/etc/fstab - file system information table
/etc/mtab - mount table

SEE ALSO

mount(2), fstab(5), mtab(5)
ULTRIX-11 System Management Guide, Sections 4.5.6 and 4.5.7

RESTRICTIONS

Mounting file systems full of garbage will crash the system. Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

mv - move or rename files and directories

SYNOPSIS

mv file1 file2

mv file ... directory

DESCRIPTION

Mv moves (changes the name of) file1 to file2.

If file2 already exists, it is removed before file1 is moved. If file2 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; if not, mv exits.

In the second form, one or more files are moved to the directory with their original file-names.

Mv refuses to move a file onto itself.

SEE ALSO

cp(1), chmod(2)

RESTRICTIONS

If file1 and file2 lie on different file systems, 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 should take -f flag, like rm, to suppress the question if the target exists and is not writable.

NAME

ncheck - generate names from i-numbers

SYNOPSIS

ncheck [-i numbers] [-a] [-s] filesystem

DESCRIPTION

Ncheck with no other arguments generates a pathname vs. i-number list of all files on the file system. Names of directory files are followed by '/.'. The -i option reduces the report to only those files whose i-numbers follow. The -a option allows printing of the names '.' and '..', which are ordinarily suppressed. suppressed. The -s option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system must be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

dcheck(1), icheck(1), sort(1)

DIAGNOSTICS

When the filesystem structure is improper, '??' denotes the 'parent' of a parentless file and a pathname beginning with '...' denotes a loop.

NAME

newgrp - log in to a new group

SYNOPSIS

newgrp group

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to login(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

A password is demanded if the group has a password and the user himself does not.

When most users log in, they are members of the group named 'other.' Newgrp is known to the shell, which executes it directly without a fork.

FILES

/etc/group, /etc/passwd

SEE ALSO

login(1), group(5)

NICE(1)

NAME

nice, nohup - run a command at low priority

SYNOPSIS

nice [-number] command [arguments]

nohup command [arguments]

DESCRIPTION

Nice executes command with low scheduling priority. If the number argument is present, the priority is incremented (higher numbers mean lower priorities) 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 priority, e.g. '--10'.

Nohup executes command immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. Nohup should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal.

FILES

nohup.out standard output and standard error file under nohup

SEE ALSO

nice(2)

DIAGNOSTICS

Nice returns the exit status of the subject command.

NAME

nm - print name list

SYNOPSIS

nm [-gnopru] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object file in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no file is given, the symbols in 'a.out' are listed.

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), B (bss segment symbol), or C (common symbol). If the symbol is local (non-external) the type letter is in lower case. If the 'a.out' file is an overlay text executable (0430 or 0431), all test segment symbols are followed by their overlay text segment number. The output is sorted alphabetically.

Options are:

-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 Don't sort; print in symbol-table order.

-r Sort in reverse order.

-u Print only undefined symbols.

SEE ALSO

ar(1), ar(5), a.out(5)

OD(1)

NAME

od - octal dump

SYNOPSIS

od [-bcdox] [file] [[+]offset[.][b]]

DESCRIPTION

Od dumps file in one or more formats as selected by the first argument. If the first argument is missing, -o is default. The meanings of the format argument characters are:

b Interpret bytes in octal.

c Interpret bytes in ASCII. 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.

d Interpret words in decimal.

o Interpret words in octal.

x Interpret words in hex.

The file argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If '.' is appended, the offset is interpreted in decimal. If 'b' is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded '+'.

Dumping continues until end-of-file.

SEE ALSO

adb(1)

NAME

osload - load optional software on Micro/PDP-11

SYNOPSIS

osload

DESCRIPTION

The osload command is used to load optional software from RX50 diskettes onto the RD51/RD52 winchester disk of the Micro/PDP-11. Refer to Section 3.8 of the ULTRIX-11 Installation Guide for a description of the osload command.

PASSWD(1)

NAME

passwd - change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the user name (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. 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; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, Password Security: A Case History

NAME

paste - merge same lines of several files or subsequent lines of one file

SYNOPSIS

```
paste file1 file2 ...
paste -dlist file1 file2 ...
paste -s [-dlist] file1 file2 ...
```

DESCRIPTION

In the first two forms, paste concatenates corresponding lines of the given input files file1, file2, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of cat(1) which concatenates vertically, i.e., one file after the other. In the last form above, paste subsumes 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. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if - is used in place of a file name.

The meanings of the options are:

-d Without this option, the new-line 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 (see below).

list One or more characters immediately following -d replace the default tab as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no -s option), the lines from the last file are always terminated with a new-line character, not from the list. The list may contain the special escape sequences: \n (new-line), \t (tab), \\ (backslash), and \0 (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use "" -d"\\\\").

-s Merge subsequent lines rather than one from each input file. Use tab for concatenation, unless a list is specified with -d option. Regardless of the list, the very last character of the file is forced to be a new-line.

- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

PASTE(1)

EXAMPLES

```
ls | paste -d" " -
                    list directory in one column

ls | paste - - - -
                    list directory in four columns

paste -s -d"\t\n" file
                    combine pairs of lines into lines
```

SEE ALSO

grep(1), cut(1),
pr(1): pr -t -m... works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

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

plot - graphics filters

SYNOPSIS

plot [-Tterminal [raster]]

DESCRIPTION

These commands read plotting instructions (see plot(5)) from the standard input, and in general produce plotting instructions suitable for a particular terminal on the standard output.

If no terminal type is specified, the environment parameter $TERM (see environ(5)) is used. Known terminals are:

4014 Tektronix 4014 storage scope.

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

ver Versatec D1200A printer-plotter. This version of plot places a scan-converted image in '/usr/tmp/raster' and sends the result directly to the plotter device rather than to the standard output. The optional argument causes a previously scan-converted file raster to be sent to the plotter.

FILES

/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/tmp/raster

SEE ALSO

plot(3), plot(5)

RESTRICTIONS

There is no lockout protection for /usr/tmp/raster.

PR(1)

NAME

pr - print file

SYNOPSIS

pr [option] ... [file] ...

DESCRIPTION

Pr produces a printed listing of one or more files. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, pr prints its standard input.

Options apply to all following files but may be reset between files:

-n Produce n-column output.

+n Begin printing with page n.

-h Take the next argument as a page header.

-wn For purposes of multi-column output, take the width of the page to be n characters instead of the default 72.

-ln Take the length of the page to be n lines instead of the default 66.

-t Do not print the 5-line header or the 5-line trailer normally supplied for each page.

-sc 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.

-m Print all files simultaneously, each in one column,

Inter-terminal messages via write(1) are forbidden during a pr.

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(1)

DIAGNOSTICS

There are no diagnostics when pr is printing on a terminal.

NAME

prep - prepare text for statistical processing

SYNOPSIS

prep [-dio] file ...

DESCRIPTION

Prep reads each file in sequence and writes it on the standard output, one 'word' to a line. A word is a string of alphabetic characters and imbedded apostrophes, delimited by space or punctuation. Hyphenated words are broken apart; hyphens at the end of lines are removed and the hyphenated parts are joined. Strings of digits are discarded.

The following option letters may appear in any order:

-d Print the word number (in the input stream) with each word.

-i Take the next file as an 'ignore' file. These words will not appear in the output. (They will be counted, for purposes of the -d count.)

-o Take the next file as an 'only' file. Only these words will appear in the output. (All other words will also be counted for the -d count.)

-p Include punctuation marks (single nonalphanumeric characters) as separate output lines. The punctuation marks are not counted for the -d count.

Ignore and only files contain words, one per line.

SEE ALSO

deroff(1)

PRINTENV(1)

NAME

printenv - print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a name is specified, only its value is printed.

If a name is specified and it is not defined in the environment, printenv returns exit status 1, else it returns status 0.

SEE ALSO

csh(1), sh(1), environ(5)

NAME

prof - display profile data

SYNOPSIS

prof [-v] [-a] [-l] [-low [-high]] [file]

DESCRIPTION

Prof interprets the file mon.out produced by the monitor subroutine. Under default modes, the symbol table in the named object file (a.out default) is read and correlated with the mon.out profile file. 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 the -a option is used, all symbols are reported rather than just external symbols. If the -l option is used, the output is listed by symbol value rather than decreasing percentage.

If the -v option is used, all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the plot(1) filters. The numbers low and high, by default 0 and 100, cause a selected percentage of the profile to be plotted with accordingly higher resolution.

In order for the number of calls to a routine to be tallied, the -p option of cc must have been given when the file containing the routine was compiled. This option also arranges for the mon.out file to be produced automatically.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

monitor(3), profil(2), cc(1), plot(1)

RESTRICTIONS

Beware of quantization errors.

NAME

prs - print an SCCS file

SYNOPSIS

prs [-d[dataspec]] [-r[SID]] [-e] [-l] [-a] files

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; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to prs, which may appear in any order, consist of keyletter arguments, and file names.

All the described keyletter arguments apply independently to each named file:

-d[dataspec] Used to 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] Used to 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.

-e Requests information for all deltas created earlier than and including the delta designated via the -r keyletter.

-l Requests information for all deltas created later than and including the delta designated via the -r keyletter.

-a Requests printing of information for both removed, i.e., delta type = R, (see rmdel(1)) and existing, i.e., delta type = D, deltas. If the -a keyletter is not specified, information for existing deltas only is provided.

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 carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line 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

PRS(1)

: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(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of what(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	what(1) 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

```
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
```

```
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:

```
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 keyletter argument allowed to be used with the special case is the -a keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use help(1) for explanations.

PS(1)

NAME

ps - process status

SYNOPSIS

ps [alxvt#] [namelist] [corefile]

DESCRIPTION

Ps prints certain indicia about active processes. The a option asks for information about all processes with terminals (ordinarily only one's own processes are displayed); x asks even about processes with no terminal; l asks for a long listing. The short listing contains the process ID, tty letter, the cumulative execution time of the process and an approximation to the command line. If v is given and l is not present, the sums of the child process's system and user times are printed following the cumulative execution time. The t option limits printouts to those processes associated with tty #. Specifying ? as the tty number to the t option will limit printouts to those processes not associated with a tty.

The long listing is columnar and contains

F Flags associated with the process. 001: in core; 002: system process; 004: locked in core (e.g. for physical I/O); 010: being swapped; 020: being traced by another process; 040: another tracing flag; 100: user settable lock in core; 200: detached inheritaed by init; 400: using new signal mechanism.

S The state of the process. 0: nonexistent; S: sleeping; W: waiting; R: running; I: intermediate; Z: terminated; T: stopped.

UID The user ID of the process owner.

PID The process ID of the process; as in certain cults it is possible to kill a process if you know its true name.

PPID The process ID of the parent process.

CPU Processor utilization for scheduling.

PRI The priority of the process; high numbers mean low priority.

NICE Used in priority computation.

ADDR If the process is resident, the memory address in octal expressed as 64 byte clicks. ADDR is left shifted six places to obtain the physical memory address. If the

process is swapped out, the disk address in decimal, that is, the block number relative to the start of the swap area, where the image of the process resides.

SZ The size in 512 byte blocks of the core image of the process.

WCHAN
The event for which the process is waiting or sleeping; if blank, the process is running.

TTY The controlling tty for the process.

TIME The cumulative execution time for the process.

The command and its arguments.

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>. Ps makes an educated guess as to the file name and arguments given when the process was created by examining core 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.

The optional third argument specifies a corefile to be used in place of /dev/mem . This is used for postmortem system debugging. If a second argument is given, it is taken to be the file containing the system's namelist.

FILES

/unix	default system namelist
/dev/mem	default core memory file
/usr/crash/core	alternate core file for crash dump analysis
/dev	searched to find swap device and tty names

SEE ALSO
kill(1)
ULTRIX-11 System Management Guide, Section 9.5.6

RESTRICTIONS
Things can change while ps is running; the picture it gives is only a close approximation to reality.
Some data printed for defunct processes is irrelevant

PSTAT(1M)

NAME

pstat - print system facts

SYNOPSIS

pstat [-aixptuf] [suboptions] [corefile] [namelist]

DESCRIPTION

Pstat interprets the contents of certain system tables. If corefile is given, the tables are sought there, otherwise in /dev/mem. The required namelist is taken from /unix , unless the optional namelist argument is given. If the namelist is specified then the corefile must also be specified. Options are

-a Under -p, describe all process slots rather than just active ones.

-i Print the inode table with the these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- L locked
- U update time filsys(5)) must be corrected
- A access time must be corrected
- M file system is mounted here
- W wanted by another process (L flag is on)
- T contains a text file
- C changed time must be corrected

CNT Number of open file table entries for this inode.
DEV Major and minor device number of file system in which this inode resides.
INO I-number within the device.
MODE Mode bits, see chmod(2).
NLK Number of links to this inode.
UID User ID of owner.
SIZ/DEV
Number of bytes in an ordinary file, or major and minor device of special file.

-x Print the text table with these headings:

LOC The core location of this table entry.
FLAGS Miscellaneous state variables encoded thus:
- T ptrace(2) in effect
- W text not yet written on swap device
- L loading in progress
- K locked
- w wanted (L flag is on)

DADDR Disk address in swap, measured in (decimal) multiples of 512 bytes.

CADDR Core address, measured in (octal) multiples of 64 bytes.

SIZE Size of text segment, measured in (decimal) multiples of 64 bytes.

IPTR Core location of corresponding inode.

CNT Number of processes using this text segment.

CCNT Number of processes in core using this text segment.

-p Print process table for active processes with these headings:

LOC The core location of this table entry.
S Run state encoded thus:
0 no process
1 waiting for some event
3 runnable
4 being created
5 being terminated
6 stopped under trace
F Miscellaneous state variables, or-ed together:
01 loaded
02 the scheduler process
04 locked
010 swapped out
020 traced
040 used in tracing
0100 locked in by lock(2)
0200 detached inherited by init
0400 using new signal mechanism.
PRI Scheduling priority, see nice(2).
SIGNAL
Signals received (signals 1-16 coded in bits 0-15),
UID Real user ID.
TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see nice(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
ADDR If in core, the physical address of the 'u-area' of the process measured in (octal) multiples of 64 bytes. If swapped out, the position in the swap area measured in (decimal) multiples of 512 bytes.
SIZE If in core, size of process image in (decimal) multiples of 64 bytes. If swapped out, size of the process image in (decimal) 512 byte blocks.
WCHAN Wait channel number of a waiting process.

PSTAT(1M)

LINK Link pointer in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for alarm(2) measured in seconds.

-t Print table for terminals (DH, KL, DL, DZ, and DZV handled) with these headings:

RAW Number of characters in raw input queue.
CAN Number of characters in canonicalized input queue.
OUT Number of characters in output queue.
MODE See tty(4).
ADDR Physical device address.
DEL Number of delimiters (newlines) in canonicalized input queue.
COL Calculated column position of terminal.
STATE Miscellaneous state variables encoded thus:
- T timeout (carriage return or newline delay processing)
- W waiting for open to complete
- O open
- C carrier is on
- B busy doing output
- A process is awaiting output
- X open for exclusive use
- S output is stopped by stop character
- M blocked output (tandem mode, see tty(4))
- H hangup on close

PGRP Process group for which this is controlling terminal.
ERRCNT
Number of errors on the line, since the system was last booted.
LASTEC
Last error character, as received on the line. Character in lo byte, error bits in hi byte.
Bit 15 - Combined error bit
Bit 14 - Overrun error
Bit 13 - Framing error
Bit 12 - Parity error
For DL and KL all three types of errors are counted. For DH, DHU/DHV, DZ/DZV, and DZQ, overruns are counted, framing errors are not counted, and parity errors are counted only if parity is enabled.

-u print information about a user process; the next argument is its address as given by ps(1). The process must be in main memory, or the file used can be a core image and the address 0.

-f Print the open file table with these headings:

LOC The core location of this table entry.

FLG Miscellaneous state variables encoded thus:
R open for reading
W open for writing
P pipe
CNT Number of processes that know this open file.
INO The location of the inode table entry for this file.
OFFS The file offset, see lseek(2).

FILES
/unix default namelist
/dev/mem default source of tables

SEE ALSO
ps(1), stat(2), filsys(5)
ULTRIX-11 System Management Guide, Section 9.5.6
K. Thompson, UNIX Implementation

PTX(1)

NAME

ptx - permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file input on file output (standard input and output default). It 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 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.

The following options can be applied:

-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.

The index for this manual was generated using ptx.

FILES

/bin/sort
/usr/lib/eign

RESTRICTIONS

Line length counts do not account for overstriking or proportional spacing.

PWD(1)

NAME

pwd - working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

cd(1)

NAME

quot - summarize file system ownership

SYNOPSIS

quot [option] ... [filesystem]

DESCRIPTION

Quot prints the number of blocks in the named filesystem currently owned by each user. If no filesystem is named, a default name is assumed. The default name is obtained from the file system information table (/etc/fstab) by searching for the file system associated with the /usr directory. The following options are available:

-n Cause the pipeline ncheck filesystem | sort +0n | quot -n filesystem to produce a list of all files and their owners.

-c Print 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 Print count of number of files as well as space owned by each user.

FILES

/dev/r??? - filesystem
/etc/fstab - default file system name
/etc/passwd - to get user names

SEE ALSO

ls(1), du(1)

DIAGNOSTICS

If no filesystem is given and no filesystem is mounted on the /usr directory, an error message is printed.

RESTRICTIONS

Holes in files are counted as if they actually occupied space.

RANLIB(1)

NAME

ranlib - convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

Ranlib converts each archive to a form which can be loaded more rapidly by the loader, by adding a table of contents named __.SYMDEF to the beginning of the archive. It uses ar(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system containing the current directory.

SEE ALSO

ld(1), ar(1)

RESTRICTIONS

Because generation of a library by ar and randomization by ranlib are separate, phase errors are possible. The loader ld warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

NAME

rasize - report on MSCP disk partition sizes

SYNOPSIS

rasize

DESCRIPTION

The RA disk driver supports a variety of MSCP disks. The partition sizes for MSCP disks are not known in advance, because this information is read from the disk when the driver brings it on-line.

The rasize command reports the following information about MSCP disks:

1. The controller type (UDA50, RQDX1, KLESI, or RUX1) and micro-code revision of each MSCP disk controller on the system.

2. The status of each MSCP disk unit, NED for a non-existent drive or the disk type (RC25, RA60, RA80, RA81, RX50, RD51 or RD52) if the unit exists. A disk is non-existent until it becomes available to the driver, this occurs on the first access of the disk unit.

3. Which of the eight possible disk partitions may be used, when creating file systems.

4. The absolute maximum sized file system that may be created on each available disk partition. This size number should be supplied to the /etc/mkfs command when making file systems on MSCP disk partitions.

FILES

/unix - system namelist
/dev/mem - system tables

SEE ALSO

ULTRIX-11 System Management Guide, Sections 1.3 and 4.6.3

DIAGNOSTICS

May cause a "drive offline" message on the console terminal if the RX50 drives are not online.

RATFOR(1)

NAME

ratfor - rational Fortran dialect

SYNOPSIS

ratfor [option ...] [filename ...]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. Ratfor provides control flow constructs essentially identical to those in C:

```
statement grouping:
     { statement; statement; statement }

decision-making:
     if (condition) statement [ else statement ]
     switch (integer value) {
          case integer:  statement
          ...
          [ default: ]   statement
     }

loops:
     while (condition) statement
     for (expression; condition; expression) statement
     do limits statement
     repeat statement [ until (condition) ]
     break [n]
     next [n]
```

and some syntactic sugar to make programs easier to read and write:

```
free form input:
     multiple statements/line; automatic continuation

comments:
     # this is a comment

translation of relationals:
     >, >=, etc., become .GT., .GE., etc.

return (expression)
     returns expression to caller from function

define:
     define name replacement

include:
     include filename
```

The option -h causes quoted strings to be turned into 27H constructs. -C copies comments to the output, and attempts

to format it neatly. Normally, continuation lines are marked with a & in column 1; the option -6x makes the continuation character x and places it in column 6.

Ratfor is best used with f77(1).

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, Software Tools, Addison-Wesley, 1976.

REFER(1)

NAME

refer, lookbib - find and insert literature references in documents

SYNOPSIS

refer [option] ...

lookbib [file] ...

DESCRIPTION

Lookbib accepts keywords from the standard input and searches a bibliographic data base for references that contain those keywords anywhere in title, author, journal name, etc. Matching references are printed on the standard output. Blank lines are taken as delimiters between queries.

Refer is a preprocessor for nroff or troff(1) that finds and formats references. The input files (standard input default) are copied to the standard output, except for lines between .[and .] command lines, which are assumed to contain keywords as for lookbib, and are replaced by information from the bibliographic data base. The user may avoid the search, override fields from it, or add new fields. The reference data, from whatever source, are 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.

The following options are available:

-ar Reverse the first r author names (Jones, J. A. instead of J. A. Jones). If r is omitted all author names are reversed.

-b Bare mode: do not put any flags in text (neither numbers nor labels).

-cstring
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in string.

-e Instead of leaving the references where encountered, accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

-kx Instead of numbering references, use labels as

specified in a reference data line beginning %x; by default x is L.

-lm,n Instead of numbering references, use labels made 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 m or ,n is omitted the entire name or date respectively is used.

-p Take the next argument as a file of references to be searched. The default file is searched last.

-n Do not search the default file.

-skeys
Sort references by fields whose key-letters are in the keys string; permute reference numbers in text accordingly. Implies -e. The key-letters in keys may be followed by a number to indicate how many such fields are used, with + taken as a very large number. The default is AD which sorts on the senior author and then date; to sort, for example, on all authors and then title use -sA+T.

To use your own references, put them in the format described in pubindex(1) They can be searched more rapidly by running pubindex(1) on them before using refer; failure to index results in a linear search.

When refer is used with eqn, neqn or tbl, refer should be first, to minimize the volume of data passed through pipes.

FILES
/usr/dict/papers directory of default publication lists and indexes
/usr/lib/refer directory of programs

RENICE(1)

NAME

renice - oppress running processes

SYNOPSIS

renice newnice pid1 [pid2 ... pidn]

DESCRIPTION

Renice will assign pid1 , ..., pidn a 'niceness' of newnice. Positive niceness means a lower scheduling priority. Negative niceness means a high priority. The super-user may renice any process to any value. Other users may renice their own processes to any value greater than the current value.

SEE ALSO

nice(1), nice(2), renice(2)

NAME

restor - incremental file system restore

SYNOPSIS

restor key [argument ...]

DESCRIPTION

Restor is used to read magtapes dumped with the dump command. The key specifies what is to be done. Key is one of the characters rRxtm optionally combined with f.

f Use the first argument as the name of the tape instead of the default.

r or R
The tape is read and loaded into the file system specified in argument. This should not be done lightly (see below). If the key is R restor asks which tape of a multi volume set to start on. This allows restor to be interrupted and then restarted (an icheck -s must be done before restart).

x Each file on the tape named by an argument is extracted. The file name has all 'mount' prefixes removed; for example, /usr/bin/lpr is named /bin/lpr on the tape. The file extracted is placed in a file with a numeric name supplied by restor (actually the inode number). In order to keep the amount of tape read to a minimum, the following procedure is recommended:

Mount volume 1 of the set of dump tapes.

Type the restor command.

Restor will announce whether or not it found the files, give the number it will name the file, and rewind the tape.

It then asks you to 'mount the desired tape volume'. Type the number of the volume you choose. On a multivolume dump the recommended procedure is to mount the last through the first volume in that order. Restor checks to see if any of the files requested are on the mounted tape (or a later tape, thus the reverse order) and doesn't read through the tape if no files are. If you are working with a single volume dump or the number of files being restored is large, respond to the query with '1' and restor will read the tapes in sequential order.

If you have a hierarchy to restore you can use dumpdir(1) to produce the list of names and a shell script

RESTOR(1M)

to move the resulting files to their homes.

t Print the date the tape was written and the date the filesystem was dumped from.

m Micro/pdp-11, restor from RX50 floppy diskettes instead of magtape. The 'r' option must accompany the 'm' option, for example:

```
restor rmf /dev/rrx2 /dev/rrd0
```

To allow an interrupted restor to be restarted, restor asks for the starting volume number. Before an interrupted restor may be restarted, an icheck -s must be done on the file system.

The r option should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape onto this. Thus

```
/etc/mkfs /dev/rhp03 40600
restor r /dev/rhp03
```

is a typical sequence to restore a complete dump. Another restor can be done to get an incremental dump in on top of this.

A dump followed by a mkfs and a restor is used to change the size of a file system.

FILES

/dev/rht0 - default tape unit
rst*

SEE ALSO

dump(1), mkfs(1), dumpdir(1)

DIAGNOSTICS

There are various diagnostics involved with reading the tape and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

RESTRICTIONS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, restor doesn't use it.

NAME

rev - reverse lines of a file

SYNOPSIS

rev [file] ...

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.

RM(1)

NAME

rm, rmdir - remove (unlink) files

SYNOPSIS

rm [-fri] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked when the -f (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument -r has been used. In that case, rm recursively deletes the entire contents of the specified directory, and the directory itself.

If the -i (interactive) option is in effect, rm asks whether to delete each file, and, under -r, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2)

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file '..' merely to avoid the antisocial consequences of inadvertently doing something like 'rm -r .*'.

NAME

rmdel - remove a delta from an SCCS file

SYNOPSIS

rmdel -rSID files

DESCRIPTION

Rmdel removes the delta specified by the SID from each named SCCS file. 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 specified must not be that of a version being edited for the purpose of making a delta (i. e., if a p-file (see get(1)) exists for the named SCCS file, the 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))

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use help(1) for explanations.

ROFF(1)

NAME

roff - format text

SYNOPSIS

roff [+n] [-n] [-s] [-h] file ...

nroff -mr [option] ... file ...
troff -mr [option] ... file ...

DESCRIPTION

Roff formats text according to control lines embedded in the text in the given files. Encountering a nonexistent file terminates printing. Incoming inter-terminal messages are turned off during printing. The optional flag arguments mean:

+n Start printing at the first page with number n.
-n Stop printing at the first page numbered higher than n.
-s Stop before each page (including the first) to allow paper manipulation; resume on receipt of an interrupt signal.
-h Insert tabs in the output stream to replace spaces whenever appropriate.

Input consists of intermixed text lines, which contain information to be formatted, and request lines, which contain instructions about how to format it. Request lines begin with a distinguished control character, normally a period.

Output lines may be filled as nearly as possible with words without regard to input lineation. Line breaks may be caused at specified places by certain commands, or by the appearance of an empty input line or an input line beginning with a space.

The capabilities of roff are specified in the attached Request Summary. Numerical values are denoted there by n or +n, titles by t, and single characters by c. Numbers denoted +n may be signed + or -, in which case they signify relative changes to a quantity, otherwise they signify an absolute resetting. Missing n fields are ordinarily taken to be 1, missing t fields to be empty, and c fields to shut off the appropriate special interpretation.

Running titles usually appear at top and bottom of every page. They are set by requests like

.he 'part1'part2'part3'

Part1 is left justified, part2 is centered, and part3 is right justified on the page. Any % sign in a title is replaced by the current page number. Any nonblank may serve

as a quote.

ASCII tab characters are replaced in the input by a replacement character, normally a space, according to the column settings given by a .ta command. (See .tr for how to convert this character on output.)

Automatic hyphenation of filled output is done under control of .hy. When a word contains a designated hyphenation character, that character disappears from the output and hyphens can be introduced into the word at the marked places only.

The -mr option of nroff or troff(1) simulates roff to the greatest extent possible.

FILES

/usr/lib/suftab suffix hyphenation tables
/tmp/rtm? temporary

RESTRICTIONS

Roff is the simplest of the text formatting programs, and is utterly frozen.

REQUEST SUMMARY

Request	Break	Initial	Meaning
.ad	yes	yes	Begin adjusting right margins.
.ar	no	arabic	Arabic page numbers.
.br	yes	-	Causes a line break the filling of the current line is stopped.
.bl n	yes	-	Insert of n blank lines, on new page if necessary.
.bp +n	yes	n=1	Begin new page and number it n; no n means '+1'.
.cc c	no	c=.	Control character becomes 'c'.
.ce n	yes	-	Center the next n input lines, without filling.
.de xx	no	-	Define parameterless macro to be invoked by request '.xx' (definition ends on line beginning '..').
.ds	yes	no	Double space; same as '.ls 2'.
.ef t	no	t=	Even foot title becomes t.
.eh t	no	t=	Even head title becomes t.
.fi	yes	yes	Begin filling output lines.
.fo	no	t=	All foot titles are t.
.hc c	no	none	Hyphenation character becomes 'c'.
.he t	no	t=	All head titles are t.
.hx	no	-	Title lines are suppressed.
.hy n	no	n=1	Hyphenation is done, if n=1; and is not done, if n=0.
.ig	no	-	Ignore input lines through a line

ROFF(1)

			beginning with '..'.
.in +n	yes	-	Indent n spaces from left margin.
.ix +n	no	-	Same as '.in' but without break.
.li n	no	-	Literal, treat next n lines as text.
.ll +n	no	n=65	Line length including indent is n characters.
.ls +n	yes	n=1	Line spacing set to n lines per output line.
.ml n	no	n=2	Put n blank lines between the top of page and head title.
.m2 n	no	n=2	n blank lines put between head title and beginning of text on page.
.m3 n	no	n=1	n blank lines put between end of text and foot title.
.m4 n	no	n=3	n blank lines put between the foot title and the bottom of page.
.na	yes	no	Stop adjusting the right margin.
.ne n	no	-	Begin new page, if n output lines cannot fit on present page.
.nn +n	no	-	The next n output lines are not numbered.
.nl	no	no	Add 5 to page offset; number lines in margin from 1 on each page.
.n2 n	no	no	Add 5 to page offset; number lines from n; stop if n=0.
.ni +n	no	n=0	Line numbers are indented n.
.nf	yes	no	Stop filling output lines.
.nx file	-	Switch	input to 'file'.
.of t	no	t=	Odd foot title becomes t.
.oh t	no	t=	Odd head title becomes t.
.pa +n	yes	n=1	Same as '.bp'.
.pl +n	no	n=66	Total paper length taken to be n lines.
.po +n	no	n=0	Page offset. All lines are preceded by n spaces.
.ro	no	arabic	Roman page numbers.
.sk n	no	-	Produce n blank pages starting next page.
.sp n	yes	-	Insert block of n blank lines, except at top of page.
.ss	yes	yes	Single space output lines, equivalent to '.ls 1'.
.ta n n..		-	Pseudotab settings. Initial tab settings are columns 9 17 25 ...
.tc c	no	space	Tab replacement character becomes 'c'.
.ti +n	yes	-	Temporarily indent next output line n spaces.
.tr cdef..	no	-	Translate c into d, e into f, etc.
.ul n	no	-	Underline the letters and numbers in the next n input lines.

NAME

rx2fmt - format RX02 floppy diskettes

SYNOPSIS

/etc/rx2fmt [-s] unit

DESCRIPTION

The rx2fmt command formats RX02 floppy diskettes for use in either single or double density mode. Single density mode provides RX01 compatibility. The '-s' option specifies single density, the default mode is double density. The 'unit' is the physical unit number of the RX02 drive to be used to format the diskette.

FILES

/dev/rhx#

SEE ALSO

hx(4)

NAME

sa, accton - system accounting

SYNOPSIS

sa [-abcjlnrstuv] [file]

/etc/accton [file]

DESCRIPTION

With an argument naming an existing file, 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 /usr/adm/acct into a summary file /usr/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 acct can grow by 100 blocks per day. The summary file is 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; sha is the default. There are zillions of options:

a Place all command names containing unprintable characters and those used only once 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.

j Instead of total minutes time for each category, give seconds per call.

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 /usr/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 command in the accounting file the user ID and command name.

v If the next character is a digit n, then type the name of each command used n times or fewer. Await a reply from the typewriter; if it begins with 'y', add the command to the category '**junk**.' This is used to strip out garbage.

FILES

/usr/adm/acct raw accounting
/usr/adm/savacct summary
/usr/adm/usracct per-user summary

SEE ALSO

ac(1), acct(2)

SACT(1)

NAME

sact - print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when get(1) with the -e option has been previously executed without a subsequent execution of delta(1). 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 spaces.

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 (i.e. executed a get 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), unget(1).

DIAGNOSTICS

Use help(1) for explanations.

NAME

sccsdiff - compare two versions of an SCCS file

SYNOPSIS

sccsdiff -rSID1 -rSID2 [-p] [-sn] files

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 arguments apply to all files.

-rSID? SID1 and SID2 specify the deltas of an SCCS file that are to be compared. Versions are passed to bdiff(1) in the order given.

-p pipe output for each file through pr(1).

-sn n is the file segment size that bdiff will pass to diff(1). This is useful when diff fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).
Source Code Control System User's Guideby L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

'file: No differences' If the two versions are the same. Use help(1) for explanations.

SCRIPT(1)

NAME

script - make typescript of terminal session

SYNOPSIS

script [-a] [-n] [-q] [-s] [-S shell] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is saved in a file, and can be sent to the line printer later with lpr. If file is given, the typescript is saved there. If not, the typescript is saved in the file typescript.

To exit script, type control-D. This sends an end of file to all processes you have started up, and causes script to exit. For this reason, control-D behaves as though you had typed an infinite number of control-D's.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply. The options are:

-a Append to the typescript file instead of creating a new file.

-n Use the 'new' shell (interpretation of 'new' is installation dependent).

-q Suppress the 'script started' and 'script done' messages.

-s Use the 'standard' shell (usually sh(1)).

-S Use shell. If the requested shell is not available, script uses any shell it can find.

AUTHOR

Mark Horton

RESTRICTIONS

Since UNIX has no way to write an end-of-file down a pipe without closing the pipe, there is no way to simulate a single control-D without ending script.

The new shell has its standard input coming from a pipe rather than a tty, so stty(1) will not work, and neither will ttyname(1).

When the user interrupts a printing process, script attempts to flush the output backed up in the pipe for better response. Usually the next prompt also gets flushed.

NAME

sed - stream editor

SYNOPSIS

sed [-n] [-e script] [-f sfile] [file] ...

DESCRIPTION

Sed copies the named files (standard input default) to the standard output, edited according to a script of commands. The -f option causes the script to be taken from file sfile; these options accumulate. If there is just one -e option and no -f's, the flag -e may be omitted. The -n option suppresses the default output.

A script consists 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), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to the standard output (except under -n) and deletes the pattern space.

An address is either a decimal number that counts input lines cumulatively across files, a '$' that addresses the last line of input, or a context address, '/regular expression/', in the style of ed(1) modified thus:

The escape sequence '\n' matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function '!' (below).

SED(1)

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

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 blanks and tabs against the stripping that is done on every script line.

An argument denoted rfile or wfile must terminate the command line and must be preceded by exactly one blank. Each wfile is created before processing begins. There can be at most 10 distinct wfile 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. If label is empty, branch to the end of the script.

(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 rfile
Read the contents of rfile. 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

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 wfile
Write. Append the pattern space to wfile 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 wfile
Write. Append the pattern space to wfile.

(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
Don't. 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.

(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.

SEE ALSO
ed(1), grep(1), awk(1)

NAME

sh, for, case, if, while, :, ., break, continue, cd, eval, exec, exit, export, login, newgrp, read, readonly, set, shift, times, trap, umask, wait - command language

SYNOPSIS

sh [-ceiknrstuvx] [arg] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See invocation for the meaning of arguments to the shell.

Commands.

A simple-command is a sequence of non blank words separated by blanks (a blank is a tab or a space). 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 exec(2)). The value of a simple-command is its exit status if it terminates normally or 200+status if it terminates abnormally (see signal(2) for a list of status values).

A pipeline is a sequence of one or more commands separated by |. The standard output of each command but the last is connected by a pipe(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A list is a sequence of one or more pipelines separated by ;, &, && or || and optionally terminated by ; or &. ; and & have equal precedence which is lower than that of && and ||, && and || also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding pipeline to be executed without waiting for it to finish. The symbol && (||) causes the list following to be executed only if the preceding pipeline returns a zero (non zero) value. Newlines may appear in a list, instead of semicolons, to delimit commands.

A command is either a simple-command or one of the following. The value returned by a command is that of the last simple-command executed in the command.

for name [in word ...] do list done

Each time a for command is executed name is set to the next word in the for word list If in word ... is omitted then in "$@" is assumed. Execution ends when there are no more words in the list.

SH(1)

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 file name generation.

if list then list [elif list then list] ... [else list] fi
The list following if is executed and if it returns zero the list following then is executed. Otherwise, the list following elif is executed and if its value is zero the list following then is executed. Failing that the else list is executed.

while list [do list] done
A while command repeatedly executes the while list and if its value is zero executes the do list; otherwise the loop terminates. The value returned by a while command is that of the last executed command in the do list. 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.

The following words are only recognized as the first word of a command and when not quoted.

if then else elif fi case in esac for while until do done { }

Command substitution.
The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing newlines are removed.

Parameter substitution.
The character $ is used to introduce substitutable parameters. Positional parameters may be assigned values by set. Variables may be set by writing

name=value [name=value] ...

${parameter}
A parameter is a sequence of letters, digits or underscores (a name), a digit, or any of the characters * @ # ? - $!. 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 a digit then it is a positional parameter. If parameter is * or @ then all the positional parameters, starting with $1, are substituted separated by spaces. $0 is set from argument zero when the shell is invoked.

${parameter-word}
If parameter is set then substitute its value; otherwise substitute word.

${parameter=word}
If parameter is not set then set it to word; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${parameter?word}
If parameter is set then substitute its value; otherwise, print word and exit from the shell. If word is omitted then a standard message is printed.

${parameter+word}
If parameter is set then substitute word; otherwise substitute nothing.

In the above word is not evaluated unless it is to be used as the substituted string. (So that, for example, echo ${d-`pwd`} will only execute pwd if d is unset.)

The following parameters are automatically set by the shell.

#	The number of positional parameters in decimal.
-	Options supplied to the shell on invocation or by set.
?	The value returned by the last executed command in decimal.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used but not set by the shell.

HOME	The default argument (home directory) for the cd command.
PATH	The search path for commands (see execution).
MAIL	If this variable is set to the name of a mail file then the shell informs the user of the arrival of mail in the specified file.
PS1	Primary prompt string, by default '$ '.
PS2	Secondary prompt string, by default '> '.
IFS	Internal field separators, normally space, tab, and newline.

Blank interpretation.
After parameter and command substitution, any 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.

File name generation.
Following substitution, each command word is scanned for the characters *, ? and [. If one of these characters appears then the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern then the word is left unchanged. The character . at the start of a file name or immediately following a /, and the character /, must be matched explicitly.

* Matches any string, including the null string.
? Matches any single character.
[...]
Matches any one of the characters enclosed. A pair of characters separated by - matches any character lexically between the pair.

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 by preceding it with a \. \newline is ignored. All characters enclosed between a pair of quote marks (''), except a single quote, are quoted. Inside double quotes ("") parameter and command substitution occurs and \ quotes the characters \ ' " and $.

"$*" is equivalent to "$1 $2 ..." whereas
"$@" is equivalent to "$1" "$2" ... .

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 then the secondary prompt ($PS2) is issued.

Input output.
Before a command is executed its 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. 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 then it is created; otherwise it is truncated to zero length.

>>word
Use file word as standard output. If the file exists then output is appended (by seeking to the end); otherwise the file is created.

<<word
The shell input is read up to a line the same as word, or end of file. The resulting document becomes the standard input. If any character of word is quoted then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, \newline is ignored, and \ is used to quote the characters \ $ ' and the first character of word.

<&digit
The standard input is duplicated from file descriptor digit; see dup(2). Similarly for the standard output using >.

<&- The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example,

```
... 2>&1
```

creates file descriptor 2 to be a duplicate of file descriptor 1.

If a command is followed by & then 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.

Environment.
The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument

list; see exec(2) and environ(5). 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. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the export command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, 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 these two lines are equivalent

```
TERM=450 cmd args
(export TERM; TERM=450; cmd args)
```

If the -k flag is set, all keyword arguments are placed in the environment, even if the occur after the command name. The following 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. (But see also trap.)

Execution.
Each time a command is executed the above substitutions are carried out. Except for the 'special commands' listed below a new process is created and an attempt is made to execute the command via an exec(2).

The shell parameter $PATH defines the search path for the directory containing the command. Each alternative directory name is separated by a colon (:). The default path is :/bin:/usr/bin. If the command name contains a / then 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 a.out file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands.
The following commands are executed in the shell process and

except where specified no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. file
Read and execute commands from file and return. The search path $PATH is used to find the directory containing file.

break [n]
Exit from the enclosing for or while loop, if any. If n is specified then break n levels.

continue [n]
Resume the next iteration of the enclosing for or while loop. If n is specified then resume at the n-th enclosing loop.

cd [arg]
Change the current directory to arg. The shell parameter $HOME is the default arg.

eval [arg ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [arg ...]
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 cause the shell input output to be modified.

exit [n]
Causes a non interactive shell to exit with the exit status specified by n. If n is omitted then the exit status is that of the last command executed. (An end of file will also exit from the shell.)

export [name ...]
The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given then a list of exportable names is printed.

login [arg ...]
Equivalent to 'exec login arg ...'.

newgrp [arg ...]
Equivalent to 'exec newgrp arg ...'.

read name ...
One line is read from the standard input; successive words of the input are assigned to the variables name in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

readonly [name ...]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given then a list of all readonly names is printed.

set [-eknptuvx [arg ...]]

-e If non interactive then exit immediately if a command fails.

-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.

- Turn off the -x and -v options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in $-.

Remaining arguments are positional parameters and are assigned, in order, to $1, $2, etc. If no arguments are given then the values of all names are printed.

shift

The positional parameters from $2... are renamed $1...

times

Print the accumulated user and system times for processes run from the shell.

trap [arg] [n] ...

Arg is a command to be read and executed when the shell receives signal(s) n. (Note that 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. If arg is absent then all trap(s) n are reset to their original values. If arg is the null string then this signal is ignored by the shell and by invoked commands. If n is 0 then the command arg is executed on exit from the shell, otherwise upon receipt of signal n as numbered in signal(2). Trap with no arguments prints a list of commands associated with each signal number.

umask [nnn]

The user file creation mask is set to the octal value nnn (see umask(2)). If nnn is omitted, the current value of the mask is printed.

wait [n]

Wait for the specified process and report its termination status. If n is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

Invocation.
If the first character of argument zero is -, commands are read from $HOME/.profile, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

-c string If the -c flag is present then commands are read from string.

-s If the -s flag is present or if no arguments remain then commands are read from the standard input. Shell output is written to file descriptor 2.

-i If the -i flag is present or if the shell input and output are attached to a terminal (as told by gtty) then this shell is interactive. In this case the terminate signal SIGTERM (see signal(2)) is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that wait is interruptable). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are described under the set command.

FILES

$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

test(1), exec(2),

DIAGNOSTICS

Errors detected by the shell, such as syntax errors cause the shell to return a non zero exit status. If the shell is being used non interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also exit).

RESTRICTIONS

If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document. A garbage file /tmp/sh* is created, and the shell complains about not being able to find the file by another name.

SIZE(1)

NAME

size - size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. For overlay text files (0430 or 0431) size prints the (decimal) number of bytes contained in the root text segment, each overlay text segment, the data and bss segments, the sum of the root, data, and bss segments, and the total text size. If no file is specified, a.out is used.

SEE ALSO

a.out(5)

NAME

sleep - suspend execution for an 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

alarm(2), sleep(3)

RESTRICTIONS

Time must be less than 65536 seconds.

SORT(1)

NAME

sort - sort or merge files

SYNOPSIS

sort [-mubdfinrtx] [+pos1 [-pos2]] ... [-o name] [-T directory] [name] ...

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name '-' means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d 'Dictionary' order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option n implies option b.

r Reverse the sense of comparisons.

tx 'Tab character' separating fields is x.

The notation +pos1 -pos2 restricts a sort key to a field beginning at pos1 and ending just before pos2. Pos1 and pos2 each have the form m.n, optionally followed by one or more of the flags bdfinr, where m tells a number of fields to skip from the beginning of the line and n tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the b option is in effect n is counted from the first nonblank in the field; b is attached independently to pos2. A missing .n means .0; a missing -pos2 means the end of the line. Under the -tx option, fields are strings separated by x; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

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.

o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

T The next argument is the name of a directory in which temporary files should be made.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

Examples. Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort -u +0f +0 list
```

Print the password file (passwd(5)) sorted by user id number (the 3rd colon-separated field).

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options -um with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um +0 -1 dates
```

FILES

/usr/tmp/stm*, /tmp/*: first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1)

SORT(1)

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option -c.

RESTRICTIONS

Very long lines are silently truncated.

NAME

spell, spellin, spellout - find spelling errors

SYNOPSIS

spell [option] ... [file] ...

/usr/lib/spellin [list]

/usr/lib/spellout [-d] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most troff, tbl and eqn(1) constructions.

Under the -v option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the -b option, British spelling is checked. Besides preferring centre, colour, speciality, travelled, etc., this option insists upon -ise in words like standardise, Fowler and the OED to the contrary notwithstanding.

Under the -x option, every plausible stem is printed with '=' for each word.

The 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.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g. thier=thy-y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by spell. Both expect a list of words, one per line, from the standard input. Spellin adds the words on the standard input to the preexisting list and places a new list on the standard output. If no list is specified, the new list is created from scratch. Spellout looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option -d) the hash list.

SPELL(1)

To create the basic hashed spelling list:
cd /usr/lib; spellin < /usr/dict/words >hlist
To create the American hashed spelling list:
cd /usr/lib; (cat american local)|spellin hlist >hlista
To create the British hashed spelling list:
cd /usr/lib; (cat british local)|spellin hlist >hlistb

FILES

SPELL_D=/usr/dict/hlist[ab]: hashed spelling lists, American & British
SPELL_S=/usr/dict/hstop: hashed stop list
SPELL_H=/usr/dict/spellhist: history file
/usr/dict/words: full dictionary
/usr/dict/american: american spelling
/usr/dict/british: british spelling
/usr/dict/local: local spelling
/usr/lib/spell
/usr/lib/spellin
/usr/lib/spellout
deroff(1), sort(1), tee(1), sed(1)

Note: spell searches the user's environment for variables of the form 'SHELL_X', if found, it uses those by default.

RESTRICTIONS

The spelling list's coverage is uneven; new installations will probably 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 [option] ...

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(1).

The following options are recognized, each as a separate argument.

-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

(2nd deriv. at end) = k*(2nd deriv. next to end)

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, i.e. 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(1)

DIAGNOSTICS

When data is not strictly monotone in x, spline reproduces the input without interpolating extra points.

RESTRICTIONS

A limit of 1000 input points is enforced silently.

SPLIT(1)

NAME

split - split a file into pieces

SYNOPSIS

split [-n] [file [name]]

DESCRIPTION

Split reads file and writes it in n-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is name with aa appended, and so on lexicographically. If no output name is given, x is default.

If no input file is given, or if - is given in its stead, then the standard input file is used.

NAME

strings - find the printable strings in a object, or other binary, file

SYNOPSIS

strings [-] [-o] [-number] file ...

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. Unless the - flag is given, strings only looks in the initialized data space of object files. If the -o flag is given, then each string is preceded by its offset in the file (in octal). If the -number flag is given then number is used as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

SEE ALSO

od(1)

AUTHOR

Bill Joy

RESTRICTIONS

The algorithm for identifying strings is extremely primitive.

NAME

strip - remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. 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.

Strip automatically accommodates overlay text files (0430 or 0431).

FILES

/tmp/stm? temporary file

SEE ALSO

ld(1)

NAME

struct - structure Fortran programs

SYNOPSIS

struct [option] ... file

DESCRIPTION

Struct translates the Fortran program specified by file (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original Fortran. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (.e.g. '.GT.' into '>'). The output is appropriately indented.

The following options may occur in any order.

-s Input is accepted in standard format, i.e. comments are specified by a c, C, or * in column 1, and continuation lines are specified by a nonzero, nonblank character in column 6. Normally, a statement whose first nonblank character is not alphanumeric is treated as a continuation.

-i Do not turn computed goto statements into switches. (Ratfor does not turn switches back into computed goto statements.)

-a Turn sequences of else ifs into a non-Ratfor switch of the form

```
switch {
       case pred1: code
       case pred2: code
       case pred3: code
       default: code
     }
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

-b Generate goto's instead of multilevel break statements.

-n Generate goto's instead of multilevel next statements.

-en If n is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If n is nonzero, admit code segments with fewer than n

statements to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop.

FILES

/tmp/struct*
/usr/lib/struct/*

SEE ALSO

f77(1)

RESTRICTIONS

Struct knows Fortran 66 syntax, but not full Fortran 77 (alternate returns, IF...THEN...ELSE, etc.)
If an input Fortran program contains identifiers which are reserved words in Ratfor, the structured version of the program will not be a valid Ratfor program.
Extended range DO's generate cryptic errors.
Columns 73-80 are not special even when -s is in effect.
Will not generate Ratfor FOR statements.

NAME

stty - set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, placing its output on the diagnostic output. With no argument, it reports the speed of the terminal and the settings of the options which are different from their defaults. With the argument 'all', all normally used option settings are reported. With the argument 'everything', everything stty knows about is printed. The option strings are selected from the following set:

even	allow even parity input
-even	disallow even parity input
odd	allow odd parity input
-odd	disallow odd parity input
raw	raw mode input (no input processing (erase, kill, interrupt, ...); parity bit passed back)
-raw	negate raw mode
cooked	same as '-raw'
cbreak	make each character available to read(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed
-cbreak	make characters available to read only when new-line is received
-nl	allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl	accept only new-line to end lines
echo	echo back every character typed
-echo	do not echo characters
lcase	map upper case to lower case
-lcase	do not map case
tandem	enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
-tandem	disable flow control
-tabs	replace tabs by spaces when printing
tabs	preserve tabs
ek	set erase and kill characters to # and @

For the following commands which take a character argument c, you may also specify c as the 'u' or 'undef', to set the value to be undefined. A value of '^x', a 2 character sequence, is also interpreted as a control character, with '^?' representing delete.

erase c — set erase character to c (default '#', but often reset to ^H.)

kill c — set kill character to c (default '@', but often reset to ^U.)

intr c — set interrupt character to c (default DEL or ^? (delete), but often reset to ^C.)

quit c — set quit character to c (default control \.)

start c — set start character to c (default control Q.)

stop c — set stop character to c (default control S.)

eof c — set end of file character to c (default control D.)

brk c — set break character to c (default undefined.) This character is an extra wakeup causing character.

cr0 cr1 cr2 cr3
select style of delay for carriage return (see ioctl(2))

nl0 nl1 nl2 nl3
select style of delay for linefeed

tab0 tab1 tab2 tab3
select style of delay for tab

ff0 ff1 — select style of delay for form feed

bs0 bs1 — select style of delay for backspace

tty33 — set all modes suitable for the Teletype Corporation Model 33 terminal.

tty37 — set all modes suitable for the Teletype Corporation Model 37 terminal.

vt05 — set all modes suitable for Digital Equipment Corp. VT05 terminal

dec — set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and 'newcrt'.)

tn300 — set all modes suitable for a General Electric TermiNet 300

ti700 — set all modes suitable for Texas Instruments 700 series terminal

tek — set all modes suitable for Tektronix 4014 terminal

0 — hang up phone line immediately

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

A teletype driver which supports the job control processing of csh(1) with more functionality than the basic driver is introduced in newtty(4) and fully described in tty(4). The following options apply only to it.

new — Use new driver (switching flushes typeahead).

old — Use old driver (switching flushes typeahead).

crt	Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)
crtbs	Echo backspaces on erase characters.
prterase	For printing terminal echo erased characters backwards within '\' and '/'.
crterase	Wipe out erased characters with 'backspace-space-backspace.'
-crterase	Leave erased characters visible; just backspace.
crtkill	Wipe out input on like kill ala crterase.
-crtkill	Just echo line kill character and a newline on line kill.
ctlecho	Echo control characters as '^x' (and delete as '^?'.) Print two backspaces following the EOT character (control D).
-ctlecho	Control characters echo as themselves; in cooked mode EOT (control-D) is not echoed.
decctlq	After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.
-decctlq	After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)
tostop	Background jobs stop if they attempt terminal output.
-tostop	Output from background jobs to the terminal is allowed.
tilde	Convert '~' to '`' on output (for Hazeltine terminals).
-tilde	Leave poor '~' alone.
flusho	Output is being discarded usually because user hit control O (internal state bit).
-flusho	Output is not being discarded.
pendin	Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).
-pendin	Input is not pending.
intrup	Send a signal (SIGTINT) to the terminal control process group whenever an input record (line in cooked mode, character in cbreak or raw mode) is available for reading.
-intrup	Don't send input available interrupts.
mdmbuf	Start/stop output on carrier transitions (not implemented).
-mdmbuf	Return error if write attempted after carrier drops.
litout	Send output characters without any processing.
-litout	Do normal output processing, inserting delays,

etc.

etxack Diablo style etx/ack handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed.

susp c set suspend process character to c (default control Z.)

dsusp c set delayed suspend process character to c (default control Y.)

rprnt c set reprint line character to c (default control R.)

flush c set flush output character to c (default control O.)

werase c set word erase character to c (default control W.)

lnext c set literal next character to c (default control V.)

SEE ALSO

tabs(1), tset(1), ioctl(2), newtty(4), tty(4)

NAME

su - substitute user id temporarily

SYNTAX

su [-] [userid]

DESCRIPTION

Su demands the password of the specified userid, and if it is given, changes to that userid and invokes a shell without changing the current directory or the user environment. The new user ID stays in force until the shell exits. If the 'SHELL' environment variable is set, it is used as the name of the new shell, and, if necessary, the previous user environment is restored for that shell. This allows for the use of multiple shells with su. If the optional - flag is given, the shell of the specified user will be the new shell, the environment is set as for that user, and the new shell is started in the new home directory. This can be used to simulate the login environment of a specified user.

If no userid is specified, 'root' is assumed. If the file /usr/adm/sulog exists, records are kept there of all su attempts to the root id. To remind the super-user of his responsibilities, the shell substitutes '#' for its usual prompt.

FILES

/usr/adm/sulog

SEE ALSO

csh(1), newgrp(1), sh(1)

SUM(1)

NAME

sum - sum and count blocks in a file

SYNOPSIS

sum file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinguishable from end of file on most devices; check the block count.

NAME

sync - update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the sync system primitive. If the system is to be stopped, sync must be called to insure file system integrity. See sync(2) for details.

SEE ALSO

sync(2), update(8)

SYSGEN(1M)

NAME

sysgen - ULTRIX-11 system generation

SYNOPSIS

sysgen

DESCRIPTION

Refer to the ULTRIX-11 System Management Guide, Chapter 2 for a description of the system generation process.

SEE ALSO

mkconf(1)

NAME

tabs - set terminal tabs

SYNOPSIS

tabs [-n] [terminal]

DESCRIPTION

Tabs sets the tabs on a variety of terminals. Various of the terminal names given in term(7) are recognized; the default is, however, suitable for most 300 baud terminals. If the -n flag is present then the left margin is not indented as is normal.

SEE ALSO

stty(1), term(7)

TAIL(1)

NAME

tail - deliver the last part of a file

SYNOPSIS

tail +number[lbc][rf] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +number from the beginning, or -number from the end of the input. 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.

Specifying r causes tail to print lines from the end of the file in reverse order. The default for r is to print the entire file this way. Specifying f causes tail to not quit at end of file, but rather wait and try to read repeatedly in hopes that the file will grow.

SEE ALSO

dd(1)

RESTRICTIONS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length.

Various kinds of anomalous behavior may happen with character special files.

NAME

tar - tape archiver

SYNOPSIS

tar [key] [name ...]

DESCRIPTION

Tar saves and restores files on magtape or RX50 floppy diskette. Its actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory. Tar also saves and restores special files.

The function portion of the key is specified by one of the following letters:

r The named files are written on the end of the tape. The c function implies this. Writing files on the end of a tar tape creates a second archive on that tape. Using the r option with a tape where the second archive already exists will overwrite the second archive, not create a third archive. Dealing with the second archive can be clumsy, i.e., the no rewind on close tape special file must be used to position the tape after the first archive. For example:

tar tf /dev/nrht0 >/dev/null

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape. The files are appended to the end of the tape (see r above).

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies r.

The following characters may be used in addition to the letter which selects the function desired.

0,...,7 This modifier selects the drive on which the tape is mounted. The default is 0.

v Normally tar does its work silently. The v (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the t function, v gives more information about the tape entries than just the name.

w causes tar to print the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is performed. Any other input means don't do it.

f causes tar to use the next argument as the name of the archive instead of /dev/rmt? or /dev/rht?, for example, tar could archive to an RX50 floppy diskette with the command

```
tar cf /dev/rrx? files
```

If the name of the file is '-', tar writes to standard output or reads from 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 move hierarchies with the command

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

b causes tar to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives or with raw disk archives that will never be updated, e.g., RX50 diskettes. The block size is determined automatically when reading tape and disk archives (key letters 'x' and 't'), unless both the 'b' and 'f' keys are specified. In that case the blocking factor specified by the 'b' key is used unconditionally.

l tells tar to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m tells tar to not restore the modification times. The mod time will be the time of extraction.

n Select (NRZI) 800 BPI density on the specified tape unit, /dev/rmt0 by default.

p Changes the mode of extracted files to the original mode, as written to tape. By default, the tar user's mode is passed to the file.

o Suppress the normal directory information. On output, tar normally places information specifying owner and mode of directories. This allows tar to extract the files even if the top level directory does not exist.

d Selects the RX50 diskettes as the tar medium, /dev/rrx0 by default. In addition, the blocking factor is established as 10, unless otherwise specified.

FILES

/dev/rmt?
/dev/rht?
/dev/rrx?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

RESTRICTIONS

There is no way to ask for the n-th occurrence of a file.
Tape errors are handled ungracefully.
The b option should not be used with archives that are going to be updated. The current magtape driver cannot backspace raw magtape. If the archive is on a disk file the b option should not be used at all, as updating an archive stored in this manner can destroy it. The b option may be used with raw disk archives if and only if those archives will never be updated.
The current limit on file name length is 100 characters.
Previous versions of tar will restore special files as empty regular files, and may return "cannot create" messages when extracting directory files.

NAME

tbl - format tables for nroff or troff

SYNOPSIS

tbl [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for nroff or troff(1). The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and reformatted. Details are given in the reference manual.

As an example, letting \t 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\tl644\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

If no arguments are given, tbl reads the standard input, so it may be used as a filter. When it is used with eqn or neqn the tbl command should be first, to minimize the volume of data passed through pipes.

SEE ALSO
troff(1), eqn(1)
M. E. Lesk, TBL.

NAME

tc - phototypesetter simulator

SYNOPSIS

tc [-t] [-sN] [-pL] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Graphic Systems phototypesetter (cat). The standard output of tc is intended for a Tektronix 4015 (a 4014 teminal with ASCII and APL character sets). The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary. Typical usage:

troff -t file | tc

At the end of each page tc waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command e will suppress the screen erase before the next page; sN will cause the next N pages to be skipped; and !line will send line to the shell.

The command line options are:

-t Don't wait between pages; for directing output into a file.

-sN Skip the first N pages.

-pL Set page length to L. L may include the scale factors p (points), i (inches), c (centimeters), and P (picas); default is picas.

'-l w'
Multiply the default aspect ratio, 1.5, of a displayed page by l/w.

SEE ALSO

troff(1), plot(1)

RESTRICTIONS

Font distinctions are lost.
The aspect ratio option is unbelievable.

NAME

ted - interactive /etc/ttys file editor

SYNOPSIS

ted

DESCRIPTION

Ted allows modification of the /etc/ttys file without the use of an editor. Ted insures that only valid user devices are created, removed, or otherwise modified by checking that a valid device name exists in /dev for each transaction.

There are a variety of commands available once ted is invoked. Terminals may be enabled and disabled using the 'modify' command. New entries are created using the 'create' command, entries are removed with the 'remove' command. General ted information is available at the main menu by entering 'help ted'. Additional help while using ted is available with the 'help' command.

Ted maintains an internal list of the changes made to the /etc/ttys file. Initially this list is a duplicate of the /etc/ttys file. The list is updated following each newly entered transaction. The list may be examined at any time using the 'print' command. The original /etc/ttys file complete with all of the current modifications is printed. The 'print' command lists the major and minor device numbers along with the name and mode of each terminal.

Changes are not actually written to the /etc/ttys file permanently until the user executes the 'write' command. The proposed changes are printed whereupon the user has the option of writing over the existing /etc/ttys file, or discarding the changes and starting over.

It is recommended that ted be run in single-user mode to avoid inadvertently disabling any active user terminals.

In multi-user mode ted may only be run by the superuser.

FILES

/etc/ttys, /etc/gettytab, /dev/tty?
/tmp/TED.EDIT_LOCK

SEE ALSO

getty(8), init(8), gettytab(5), ttys(5)
ULTRIX-11 System Management Guide, Section 4.7

TEE(1)

NAME

tee - pipe fitting

SYNOPSIS

tee [-i] [-a] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the files. Option -i ignores interrupts; option -a causes the output to be appended to the files rather than overwriting them.

NAME

test - condition command

SYNOPSIS

test expr

DESCRIPTION

test evaluates the expression expr, and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. test returns a non zero exit if there are no arguments.

The following primitives are used to construct expr.

-r file true if the file exists and is readable.

-w file true if the file exists and is writable.

-f file true if the file exists and is not a directory.

-d file true if the file exists and is a directory.

-s file true if the file 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.

-z s1 true if the length of string s1 is zero.

-n s1 true if the length of the string s1 is nonzero.

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. Any of the comparisons -ne, -gt, -ge, -lt, or -le may be used in place of -eq.

These primaries may be combined with the following operators:

! unary negation operator

-a binary and operator

-o binary or operator

(expr)
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
sh(1), find(1)

NAME

time - time a command

SYNOPSIS

time command

DESCRIPTION

The given command is executed; after it is complete, time prints 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 execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

The times are printed on the diagnostic output stream.

RESTRICTIONS

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tip, cu - connect to a remote system

SYNOPSIS

tip [-v] [-speed] system-name
tip [-v] [-speed] phone-number
cu phone-number [-t] [-s speed] [-a acu] [-l line] [-#]

DESCRIPTION

Tip and cu establish a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote cpu. It goes without saying that you must have a login on the machine (or equivalent) to which you wish to connect. The preferred interface is tip. The cu interface is included for those people attached to the 'call UNIX' command of version 7. This manual page describes only tip.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde ('~') appearing as the first character of a line is an escape signal; the following are recognized:

~^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 will return you to tip).

~> Copy file from local to remote. Tip prompts for the name of a local file to transmit.

~< Copy file from remote to local. Tip prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

~p from [to]
Send a file to a remote UNIX host. The put command causes the remote UNIX system to run the command string 'cat > 'to' ', while tip sends it the 'from' file. If the 'to' file isn't specified the 'from' file name is used. This command is actually a UNIX specific version of the '~>' command.

~t from [to]
Take a file from a remote UNIX host. As in the put command the 'to' file defaults to the 'from' file

name if it isn't 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 UNIX process. The command string sent to the local UNIX system is processed by the shell.

~# Send a BREAK to the remote system. For systems which don't support the necessary ioctl call the break is simulated by a sequence of line speed changes and DEL characters.

~s Set a variable (see the discussion below).

~^Z Stop tip (only available with job control).

~? Get a summary of the tilde escapes

Tip uses the file /etc/remote to find how to reach a particular system and to find out how it should operate while talking to the system; refer to remote(5) for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, e.g. 'tip -300 mds'.

When tip establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in /etc/remote.

When tip prompts for an argument (e.g. 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, will abort the dialogue and return you to the remote machine.

Tip guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by uucp(1C).

During file transfers tip provides a running count of the number of lines transferred. When using the ~> and ~< commands, the 'eofread' and 'eofwrite' variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, 'echocheck' may be set to indicate tip should synchronize with the remote system on the echo of each transmitted character.

When tip must dial a phone number to connect to a system it will print various messages indicating its actions. Tip supports the DEC DN-11 and Racal-Vadic 831 auto-call-units; the DEC DF02 and DF03, Ventel 212+, Racal-Vadic 3451, and Bizcomp 1031 and 1032 integral call unit/modems.

VARIABLES

Tip maintains a set of variables which control its operation. Some of these variable 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 set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a '?' to the end. For example '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 concatenating 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 file .tiprc in one's home directory). The -v option causes tip to display the sets as they are made. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values.

beautify
(bool) Discard unprintable characters when a session is being scripted; abbreviated be.

baudrate
(num) The baud rate at which the connection was established; abbreviated ba.

dialtimeout
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated dial.

echocheck
(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is off.

eofread
(str) The set of characters which signify and end-of-transmission during a ~< file transfer command; abbreviated eofr.

eofwrite
(str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated eofw.

eol
(str) The set of characters which indicate an end-of-line. Tip will recognize escape characters only after an end-of-line.

escape
(char) The command prefix (escape) character; abbreviated es; default value is '~'.

exceptions
(str) The set of characters which should not be discarded due to the beautification switch; abbreviated ex; default value is '\t\n\f\b'.

force
(char) The character used to force literal data transmission; abbreviated fo; default value is '^P'.

framesize
(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated fr.

host
(str) The name of the host to which you are connected; abbreviated ho.

prompt
(char) The character which indicates and end-of-line on the remote host; abbreviated pr; default value is '\n'. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on recipt of this character.

raise
(bool) Upper case mapping mode; abbreviated ra; default value is off. When this mode is enabled, all lower case letters will be mapped to upper case by tip for transmission to the remote machine.

raisechar
(char) The input character used to toggle upper case mapping mode; abbreviated rc; default value is '^A'.

record
(str) The name of the file in which a session script is recorded; abbreviated rec; default value is 'tip.record'.

script
(bool) Session scripting mode; abbreviated sc; default is off. When script is true, 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 betwee 040 and 0177). The variable exceptions is used to indicate characters which are an exception to the normal beautification rules.

tabexpand
(bool) Expand tabs to spaces during file transfers; abbreviated tab; default value is false. Each tab is expanded to 8 spaces.

verbose
(bool) Verbose mode; abbreviated verb; default is true. When verbose mode is enabled, tip prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

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.

FILES

/etc/remote	global system descriptions
/etc/phones	global phone number data base
${REMOTE}	private system descriptions
${PHONES}	private phone numbers
~/.tiprc	initialization file.
/usr/spool/uucp/LCK..*	lock file to avoid conflicts with uucp

DIAGNOSTICS
Diagnostics are, hopefully, self explanatory.

SEE ALSO
remote(5), phones(5)

RESTRICTIONS
The full set of variables is undocumented and should, probably, be paired down.

TK(1)

NAME

tk - paginator for the Tektronix 4014

SYNOPSIS

tk [-t] [-N] [-pL] [file]

DESCRIPTION

The output of tk is intended for a Tektronix 4014 terminal. Tk arranges for 66 lines to fit on the screen, divides the screen into N columns, and contributes an eight space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page tk waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command !command will send the command to the shell.

The command line options are:

-t Don't wait between pages; for directing output into a file.

-N Divide the screen into N columns and wait after the last column.

-pL Set page length to L lines.

SEE ALSO

pr(1)

NAME

touch - update date last modified of a file

SYNOPSIS

touch [-c] file ...

DESCRIPTION

Touch attempts to set the modified date of each file. This is done by reading a character from the file and writing it back.

If a file does not exist, an attempt will be made to create it unless the -c option is specified.

NAME

tp - manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or magtape. Its actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so './abc' can never be the same as '/usr/dmr/abc' even if '/usr/dmr' is the current directory. If no file argument is given, '.' is the default.

u updates the tape. u is like r, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. u is the default command if none is given.

d deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.

x extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

h Specifies 1600 BPI magtape as opposed to DECtape.

m Specifies 800 BPI magtape as opposed to DECtape.

0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, x is default; for magtape '0' is the default.

v Normally tp does its work silently. The v (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the t function, v gives more information about the tape entries than just the name.

c means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with r and u. This option is assumed with magtape since it is impossible to selectively overwrite magtape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option is known to work only with x.

w causes tp to pause before treating each file, type the indicative letter and the file name (as with v) and await the user's response. Response y means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response x means 'exit'; the tp command terminates immediately. In the x function, files previously asked about have been extracted already. With r, u, and d no change has been made to the tape.

FILES

/dev/tap?
/dev/mt?
/dev/ht?

SEE ALSO

ar(1), tar(1)

DIAGNOSTICS

Several; the non-obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

RESTRICTIONS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by tp difficult to carry to other machines; tar(1) avoids the problem.

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. Input characters found in string1 are mapped into the corresponding characters of string2. When string2 is short it is padded to the length of string1 by duplicating its last character. Any combination of the options -cds may be used: -c complements the set of characters in string1 with respect to the universe of characters whose ASCII codes are 01 through 0377 octal; -d deletes all input characters in string1; -s squeezes all strings of repeated output characters that are in string2 to single characters.

In either string the notation a-b means 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. A '\' followed by any other character stands for that character.

The following example creates a list of all the words in 'file1' one per line in 'file2', 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' <file1 >file2
```

SEE ALSO

ed(1), ascii(7)

RESTRICTIONS

Won't handle ASCII NUL in string1 or string2; always deletes NUL from input.

NAME

troff, nroff - text formatting and typesetting

SYNOPSIS

troff [option] ... [file] ...

nroff [option] ... [file] ...

DESCRIPTION

Troff formats text in the named files for printing on a Graphic Systems C/A/T phototypesetter; nroff for typewriter-like devices. Their capabilities are described in the Nroff/Troff user's manual.

If no file argument is present, the standard input is read. An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

-olist 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.

-nN Number first generated page N.

-sN 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. Troff will stop the phototypesetter every N pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

-mname Prepend the macro file /usr/lib/tmac/tmac.name to the input files.

-raN Set register a (one-character) to N.

-i Read standard input after the input files are exhausted.

-q Invoke the simultaneous input-output mode of the rd request.

Nroff only

-Tname Prepare output for specified terminal. Known names are 37 for the (default) Teletype Corporation Model 37 terminal, tn300 for the GE TermiNet 300 (or any terminal without half-line capability), 300S for the

NAME

tss - list TTY structure assignments

SYNOPSIS

/etc/tss

DESCRIPTION

Each port on any communications interface is assigned a TTY structure from the pool of TTY structures. The tss command lists the current TTY structure assignments.

The tss command can be useful when debugging a user written communications device driver.

DASI-300S, 300 for the DASI-300, and 450 for the DASI-450 (Diablo Hyterm).

-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.

Troff only

-t Direct output to the standard output instead of the phototypesetter.

-f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

-w Wait until phototypesetter is available, if currently busy.

-b Report whether the phototypesetter is busy or available. No text processing is done.

-a Send a printable ASCII approximation of the results to the standard output.

-pN Print all characters in point size N while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

-g Prepare output for a GCOS phototypesetter and direct it to the standard output (see gcat(1)).

If the file /usr/adm/tracct is writable, troff keeps phototypesetter accounting records there. The integrity of that file may be secured by making troff a 'set user-id' program.

FILES

/usr/lib/suftab	suffix hyphenation tables
/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 troff
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

SEE ALSO

J. F. Ossanna, Nroff/Troff user's manual
B. W. Kernighan, A TROFF Tutorial
eqn(1), tbl(1)
col(1), tk(1) (nroff only)
tc(1), gcat(1) (troff only)

NAME

true, false - provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. False does nothing, unsuccessfully. They are typically used in input to sh(1) such as:

```
while true
do
     command
done
```

SEE ALSO

sh(1)

DIAGNOSTICS

True has exit status zero, false nonzero.

NAME

tset, reset - set terminal modes

SYNOPSIS

tset [options] [-m [ident][test baudrate]:type] ... [type]

reset ...

DESCRIPTION

Tset sets up your terminal when you first log in to a UNIX system. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. It first determines the type of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each UNIX port is specified in the /etc/ttytype database. Type names for terminals may be found in the /etc/termcap database. If a port is not wired permanently to a specific terminal (not hardwired) it will be given an appropriate generic identifier such as dialup.

In the case where no arguments are specified, tset simply reads the terminal type out of the environment variable TERM and reinitializes the terminal. The rest of this manual concerns itself with mode and environment initialization, typically done once at login, and options used at initialization 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 which are not hardwired. These ports are identified in /etc/ttytype as dialup or plugboard or arpanet, etc. To specify what terminal type you usually use on these ports, the -m (map) option flag is followed by the appropriate port type identifier, an optional baud rate specification, and the terminal type. (The effect is to 'map' from some conditions to a terminal type, that is, to tell tset 'If I'm on this kind of port, guess that I'm on that kind of terminal'.) If more than one mapping is specified, the first applicable mapping prevails. A missing port type identifier matches all identifiers. Any of the alternate generic names given in termcap may be used for the identifier.

A baudrate is specified as with stty(1), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate test may be any combination of: >, @, <, and !; @ means 'at' and ! inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to -m within '''

characters; users of csh(1) must also put a '\' before any '!' used here.

Thus

```
tset -m 'dialup>300:adm3a' -m dialup:dw2 -m 'plugboard:?adm3a'
```

causes the terminal type to be set to an adm3a if the port in use is a dialup at a speed greater than 300 baud; to a dw2 if the port is (otherwise) a dialup (i.e. at 300 baud or less). If the type finally determined by tset begins with a question mark, the user is asked if s/he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Thus, in the above case, the user will be queried on a plugboard port as to whether they are actually using an adm3a.

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 identifier found in the /etc/ttytype database will be taken to be 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 option; using the Bourne shell, sh(1):

```
eval `tset -s options...`
```

or using the C shell, csh(1):

```
set noglob; eval `tset -s options...`
```

If you have an old csh with no eval command, do the following:

```
tset -s options > /tmp/tset$$ ; source /tmp/tset$$ ; rm /tmp/tset$$
```

In either csh case, you should probably make an alias in your .cshrc:

```
alias tset 'set noglob; eval `tset -s *`'
```

or

```
alias tset 'tset -s * > /tmp/tset$$ ; source /tmp/tset$$ ;
rm /tmp/tset$$'
```

Either of these aliases allow the command

```
tset 2621
```

to be invoked at any time from your login csh. 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.)

These commands cause tset to generate as output a sequence of shell commands which place the variables TERM and TERMCAP in the environment; see environ(5).

Once the terminal type is known, tset engages in terminal driver mode setting. 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 default erase character ('#' on standard systems), the erase character is changed to DELETE.

The options are:

-e*c* set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, in this case DELETE. The character *c* can either be typed directly, or entered using the hat notation used here.

-k*c* is similar to -e but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons). The kill character is left alone if -k is not specified. The hat notation can also be used for this option.

-n On systems with the new tty driver, specifies that the new tty driver modes should be initialized for this terminal. For a CRT, the CRTERASE and CRTKILL modes are set only if the baud rate is 1200 or greater. See newtty(4) for more detail.

-v On systems with CB/UNIX virtual terminals, normally tset sets up the tty driver modes and the environment TERMCAP variable to use the virtual terminal. The -v option turns off this processing, causing tset to not use a virtual terminal.

-I suppresses transmitting terminal initialization strings.

-Q suppresses printing the 'Erase set to' and 'Kill set to' messages.

-S If your shell is csh, tset will output the strings to be assigned to TERM and TERMCAP in the environment rather than commands for a shell. This is mainly useful to avoid the (slow) source command with an old csh.

If tset is invoked as reset, it will set cooked and echo modes, turn off cbreak and raw modes, turn on newline translation, and restore special characters to a sensible state before any terminal dependent processing is done. Any special character that is found to be NULL or '-1' is reset to it's default value.

This is most useful after a program dies leaving a terminal in a funny state. You may have to type '<LF>reset<LF>' to get it to work since <CR> may not work in this state. Often none of this will echo.

EXAMPLES

These examples all assume the Bourne shell and use the -s option. If you use csh, use one of the variations described above. Note that 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 not been included here to keep the examples small. (NOTE: some of the examples given here appear to take up more than one line, for text processing reasons. When you type in real tset commands, you must enter them entirely on one line.)

At the moment, you are on a 2621. This is suitable for typing by hand but not for a .profile, unless you are always on a 2621.

```
eval `tset -s 2621`
```

You have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttytype.

```
eval `tset -s -m dialup:h19`
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use a vt100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone elses terminal at work, so you

want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
eval `tset -s -m 'switch>1200:?vt100' -m 'switch<=1200:2621'`
```

All of the above entries will fall back on the terminal type specified in /etc/ttytype 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. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
eval `tset -s ?adm3a`
```

If the file /etc/ttytype is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
eval `tset -s -m '>1200:vt100' 2621`
```

Here is a fancy example to illustrate the power of tset and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in /etc/ttytype. You want your erase character set to control H, your kill character set to control U, and don't want tset to print the "Erase set to Backspace, Kill set to Control U" message.

```
eval `tset -e -k^U -Q -s -m 'switch<=1200:concept100' -m 'switch:?vt100'
    -m dialup:concept100 -m arpanet:dm2500`
```

FILES

/etc/ttytype	port name to terminal type mapping database
/etc/termcap	terminal capability database

SEE ALSO

csh(1), sh(1), stty(1), environ(5), termcap(5), ttytype(5)

NAME

tsort - topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input file. If no file is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

RESTRICTIONS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty - get terminal name

SYNOPSIS

tty

DESCRIPTION

Tty prints the pathname of the user's terminal.

DIAGNOSTICS

'not a tty' if the standard input file is not a terminal.

NAME

ul - do underlining

SYNOPSIS

ul [-i] [-t terminal] [name ...]

DESCRIPTION

Ul reads the named files (or 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. The -t option overrides the terminal kind specified in the environment. The file /etc/termcap is read 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(1). If the terminal cannot underline, underlining is ignored.

The -i option causes ul to indicate underlining onto by a separate line containing appropriate dashes '-'; this is useful when you want to look at the underlining which is present in an nroff output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

AUTHOR

Mark Horton wrote ul. The -i option was originally an option of the editor ex(1), then an iul command.

RESTRICTIONS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

unget - undo a previous get of an SCCS file

SYNOPSIS

unget [-rSID] [-s] [-n] files

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. Keyletter arguments apply independently to each named file.

-rSID — Uniquely identifies which delta is no longer intended. (This would have been specified by get as the 'new delta'). The use of this keyletter 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 and omitted on the command line.

-s — Suppresses the printout, on the standard output, of the intended delta's SID.

-n — Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), sact(1).

DIAGNOSTICS

Use help(1) for explanations.

NAME

uniq - report repeated lines in a file

SYNOPSIS

uniq [-udc [+n] [-n]] [input [output]]

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 that repeated lines must be adjacent in order to be found; see sort(1). If the -u flag is used, just the lines that are not repeated in the original file are output. The -d option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the -u and -d mode outputs.

The -c option supersedes -u and -d and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The n arguments specify skipping an initial portion of each line in the comparison:

-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 tabs and spaces from its neighbors.

+n The first n characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

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
     * 2.54000e+00
     / 3.93701e-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 pounds force/in2
You want: atm
     * 1.02069e+00
     / 9.79730e-01
```

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade 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 a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES
/usr/lib/units

RESTRICTIONS
Don't base your financial plans on the currency conversions.

NAME

usat - ULTRIX-11 System Acceptance Test

SYNOPSIS

usat [help] [option...]

DESCRIPTION

Usat tests and verifies the existence and correct operation of various ULTRIX-11 commands. Without an argument, usat lists the names of the tests which are available.

Usat runs each specified command using a test input file. After executing the command, it compares the output with a file containing the known, correct result. Files used for each test are located in /usr/lib/usat/test, where 'test' is the command name directory.

Multiple copies of usat may be run in the same directory without fear of collisions. An exception, however, is running simultaneous yacc tests in the same directory. Because a common output file y.tab.c is generated, usat cannot guarantee consistent results in this case.

EXAMPLE

usat c c

runs two copies, one after the other, of the C compiler test. A short C program found in /usr/lib/usat/cc generates a table of temperatures. The input file cc.in.c is compiled and executed. Its output is then compared with the known, correct output found in cc.sout. The .sout notation represents a (s)ave (out)put file. If the files fail to compare, the cc test fails, and a warning message is printed.

FILES

/usr/lib/usat/?, /tmp/???

SEE ALSO

usep(1)

NAME
users - compact list of users who are on the system

SYNOPSIS
users

DESCRIPTION
Users lists the login names of the users currently on the system in a compact, one-line format.

FILES
/etc/utmp

SEE ALSO
finger(1), who(1)

NAME

uucp, uulog - unix to unix copy

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name 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 which uucp knows about. Shell metacharacters ?*[] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~user; where user is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by 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 part of the source-file name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see chmod(2)).

The following options are interpreted by uucp.

-d Make all necessary directories for the file copy.

-c Use the source file when copying out rather than copying the file to the spool directory.

-m Send mail to the requester when the copy is complete.

Uulog maintains a summary log of uucp and uux(1) transactions in the file '/usr/spool/uucp/LOGFILE' by gathering information from partial log files named '/usr/spool/uucp/LOG.*.?'. It removes the partial log files.

The options cause uulog to print logging information:

-ssys
Print information about work involving system sys.

-uuser
Print information about work done for the specified user.

FILES
/usr/spool/uucp - spool directory
/usr/lib/uucp/* - other data and program files

SEE ALSO
uux(1), mail(1)
D. A. Nowitz, Uucp Implementation Description

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.

RESTRICTIONS
All files received by uucp will be owned by uucp.
The -m option will only work sending files or receiving a single file. (Receiving multiple files specified by special shell characters ?*[] will not activate the -m option.)

NAME

uux - unix to unix command execution

SYNOPSIS

uux [-] command-string

DESCRIPTION

Uux will gather 0 or more files from various systems, execute a command on a specified system and send 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

(1) a full pathname;

(2) a pathname preceded by ~xxx; where xxx is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

The '-' option will cause the standard input to the uux command to be the standard input to the command-string.

For 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.

Any special shell characters such as <>;| should be quoted either by quoting the entire command-string, or quoting the special characters as individual arguments.

FILES

/usr/uucp/spool - spool directory
/usr/uucp/* - other data and programs

SEE ALSO

uucp(1)
D. A. Nowitz, Uucp implementation description

WARNING

An installation may, and for security reasons generally will, limit the list of commands executable on behalf of an

UUX(1C)

incoming request from uux. Typically, a restricted site will permit little other than the receipt of mail via uux.

RESTRICTIONS

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

val - validate SCCS file

SYNOPSIS

val -
val [-s] [-rSID] [-mname] [-ytype] files

DESCRIPTION

Val determines if the specified file is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to val may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

Val has a special argument, -, which causes reading of the standard input until an end-of-file 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.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

-s — The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

-rSID — The argument value SID (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the SID is ambiguous (e. g., rl is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., rl.0 or rl.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.

-mname — The argument value name is compared with the SCCS %M% keyword in file.

-ytype — The argument value type is compared with the SCCS %Y% keyword in file.

The 8-bit code returned by val is a disjunction of the possible errors, i. e., 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 keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = can't 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 that 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 - a logical OR of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS

Use help(1) for explanations.

RESTRICTIONS

Val can process up to 50 files on a single command line. Any number above 50 will produce a core dump.

NAME

vc - version control

SYNOPSIS

vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

DESCRIPTION

The vc command copies lines from the standard input to the standard output under control of its arguments and control statements encountered in the standard input. In the process of performing the copy operation, user declared keywords may be replaced by their string value when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, except as modified by the -t keyletter (see below). The default control character is colon (:), except as modified by the -c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with ed(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The -a keyletter (see below) 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 \.

Keyletter arguments

-a Forces replacement of keywords surrounded by control characters with their assigned value in all text lines and not just in vc statements.

-t All characters from the beginning of a line up to and including the first tab character are ignored for the purpose of

detecting a control statement. If one is found, all characters up to and including the tab are discarded.

-cchar — Specifies a control character to be used in place of :.

-s — Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]
Used to declare keywords. All keywords must be declared.

:asg keyword=value
Used to assign values to keywords. An asg statement overrides the assignment for the corresponding keyword on the vc command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

:if condition
.
.
.
:end
Used to skip lines of the standard input. If the condition is true all lines between the if statement and the matching end statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening if statements and matching end statements are recognized solely for the purpose of maintaining the proper if-end matching.
The syntax of a condition is:

```
<cond>  ::= [ "not" ] <or>
<or>    ::= <and> | <and> "|" <or>
<and>   ::= <exp> | <exp> "&" <and>
<exp>   ::= "(" <or> ")" | <value> <op> <value>
<op>    ::= "=" | "!=" | "<" | ">"
<value> ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than

( ) used for logical groupings
not may only occur immediately after the if, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e. g.: 012 > 12 is false). All other operators take strings as arguments (e. g.: 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > < all of equal precedence
&
|

Parentheses may be used to alter the order of precedence.
Values must be separated from operators or parentheses by at least one blank or tab.

::text
Used for keyword replacement on lines that are copied to the standard output. 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. This action is independent of the -a keyletter.

:on

:off
Turn on or off keyword replacement on all lines.

:ctl char
Change the control character to char.

:msg message
Prints the given message on the diagnostic output.

:err message
Prints 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.

DIAGNOSTICS
Use help(1) for explanations.

EXIT CODES
0 - normal
1 - any error

NAME

vi - screen oriented (visual) display editor based on ex

SYNOPSIS

vi [-r] [+command] [-l] [-wn] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on ex(1). Ex and vi run the same code; it is possible to get to the command mode of ex from within vi and vice-versa.

The ULTRIX-11 Programmer's Guide and the Introduction to Display Editing with Vi provide full details on using vi.

FILES

See ex(1).

SEE ALSO

ex (1), edit (1), "ULTRIX-11 Programmer's Guide", "An Introduction to Display Editing with Vi".

AUTHOR

William Joy
Mark Horton added macros to visual mode and is maintaining version 3

RESTRICTIONS

Software tabs using ^T work only immediately after the autoindent.

Left and right shifts on intelligent terminals don't 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 won't be broken.

Insert/delete within a line can be slow if tabs 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.

NAME

volcopy, labelit - copy file systems with label checking

SYNOPSIS

/etc/volcopy [options] fsname special1 vol1 special2 vol2

/etc/labelit special [fsname volume [-n]]

DESCRIPTION

Volcopy makes a literal copy of the file system using a blocksize matched to the device.

Options are:

-a invoke a verification sequence requiring a positive operator response. (This is the default.)
-n do not ask before copying file systems.

Other options are used only with tapes:

-bpidensity density in bits-per-inch (i.e., 800/1600/6250),
-feetsize size of reel in feet (i.e., 1200/2400),
-reelnum beginning reel number num for a restarted copy,
-buf use double buffered I/O.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, volcopy will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives.

The fsname argument represents the mounted name (e.g.: 'root', 'usr', etc.) of the filsystem being copied.

The special is the physical disk section or tape (e.g.: /dev/rdsk15, /dev/rmt0, etc.).

The vol is the physical volume name (e.g.: pk3, t0122, etc.) and should match the external label sticker. Such label names are limited to six or fewer characters. Vol may be - to use the existing volume name.

Special1 and vol1 are the device and volume from which the copy of the file system is being extracted. Special2 and vol2 are the target device and volume.

Fsname and vol are recorded in the last 12 characters of the superblock (char fsname[6], volname[6];).

Labelit can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, labelit prints current label values. The -n option provides for initial labeling of new tapes only (this destroys previous contents).

FILES

/etc/log/filesave.log a record of file systems/volumes copied

RESTRICTIONS

Tape record sizes are determined both by density and by drive type. On DEC systems, records are 5,120 bytes long at 800 and 1600 bits-per-inch, and 25,600 bytes long at 6250 bits-per-inch.

NAME

wait - await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & 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

sh(1)

RESTRICTIONS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for.

NAME

wall - write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

'Cannot send to ...' when the open on a user's tty file fails.

NAME

wc - word count

SYNOPSIS

wc [-lwc] [name ...]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters l, w, or c.

NAME

what - identify SCCS files

SYNOPSIS

what files

DESCRIPTION

What searches the given files 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 ", >, new-line, \, or null character. For example, if the C program in file f.c contains

```
char ident[] = "@(#)identification information";
```

and f.c is compiled to yield f.o and a.out, then 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 command get(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Use help(1) for explanations.

RESTRICTIONS

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

who - who is on the system

SYNOPSIS

who [who-file] [am I]

DESCRIPTION

Who, without an argument, lists the login name, terminal name, and login time for each current UNIX user.

Without an argument, who examines the /etc/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file will be /usr/adm/wtmp, which contains a record of all the logins since it was created. Then who lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with '/dev/' suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in 'who am I' (and also 'who are you'), who tells who you are logged in as.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5)

NAME
whoami - print effective current user id

SYNOPSIS
whoami

DESCRIPTION
Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES
/etc/passwd

SEE ALSO
who (1)

AUTHOR
Bill Joy

NAME

write - write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message

Message from yourname yourttyname...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point write writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the ttyname argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the mesg command. At the outset writing is allowed. Certain commands, in particular nroff and pr(1) disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, write calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using write: when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a distinctive signal-(o) for 'over' is conventional-that the other may reply. (oo) for 'over and out' is suggested when conversation is about to be terminated.

FILES

/etc/utmp to find user
/bin/sh to execute '!'

SEE ALSO

mesg(1), who(1), mail(1)

NAME

xstr - extract strings from C programs to implement shared strings

SYNOPSIS

xstr [-c] [-] [file]

DESCRIPTION

Xstr maintains a file 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, most useful if they are also read-only.

The command

```
xstr -c name
```

will extract 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 suffices 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 its 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 can avoid remaking xs.o unless truly necessary.

FILES

strings	Data base of strings
x.c	massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	temporary file for 'xstr name' strings

SEE ALSO

mkstr(1)

AUTHOR

Bill Joy

RESTRICTIONS

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 will do.

NAME

yacc - yet another compiler-compiler

SYNOPSIS

yacc [-vd] 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 program yyparse. This program must be loaded with the lexical analyzer program, 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.

If the -v flag is given, the file y.output is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the -d flag is used, the file y.tab.h is generated with the define statements that associate the yacc-assigned 'token codes' with the user-declared 'token names'. This allows source files other than y.tab.c to access the token codes.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs
/lib/liby.a	library with default 'main' and 'yyerror'

SEE ALSO

lex(1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC - Yet Another Compiler Compiler by S. C. Johnson.

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.

NAME

zaptty - run a program without a controlling terminal.

SYNOPSIS

zaptty command

DESCRIPTION

Zaptty will disassociate itself from it's controlling terminal, and then exec the specified command. This is useful for re-starting daemons that have died for some reason, and don't want to be associated with any terminal when they are re-started.

Zaptty is restricted to the super-user.

SEE ALSO

zaptty(2), nohup(1)

NAME

intro, errno - introduction to system calls and error numbers

SYNOPSIS

#include <errno.h>

DESCRIPTION

Section 2 of this manual lists all the entries into the system. Most of these calls have an error return. An error condition is indicated by an otherwise impossible returned value. Almost always this is -1; the individual sections 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 occurred.

There is a table of messages associated with each error, and a routine for printing the message; See perror(3). The possible error numbers are not recited with each writeup in section 2, since many errors are possible for most of the calls. Here is a list of the error numbers, their names as defined in <errno.h>, and the messages available using perror.

0 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 superuser. It is also returned for attempts by ordinary users to do things allowed only to the superuser.

2 ENOENT No such file or directory
This error occurs when a file name is specified and the file should exist but doesn't, or when one of the directories in a path name does not exist.

3 ESRCH No such process
The process whose number was given to signal and ptrace does not exist, or is already dead.

4 EINTR Interrupted system call
An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system call. If execution is resumed after processing the signal, it will appear as if the interrupted system call returned this error condition.

5 EIO I/O error
Some physical I/O error occurred during a read or

INTRO(2)

write. 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 dialled in or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
An argument list longer than 5120 bytes is presented to exec.

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 (resp. write) request is made to a file that is open only for writing (resp. reading).

10 ECHILD No children
Wait and the process has no living or unwaited-for children.

11 EAGAIN No more processes
In a fork, the system's process table is full or the user is not allowed to create any more processes.

12 ENOMEM Not enough core
During an exec or break, a program asks for more core than the system is able to supply. This is not a temporary condition; the maximum core size is a system parameter. The error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers.

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 plain file was mentioned where a block device was required, e.g. in mount.

16 EBUSY Mount device busy
An attempt to mount a device that was already mounted or an attempt was made to dismount a device on which there is an active file (open file, current directory, mounted-on file, active text segment).

17 EEXIST File exists
An existing file was mentioned in an inappropriate context, e.g. link.

18 EXDEV Cross-device link
A link to a file on another device was attempted.

19 ENODEV No such device
An attempt was made to apply an inappropriate system call to a device; e.g. read a write-only device.

20 ENOTDIR Not a directory
A non-directory was specified where a directory is required, for example in a path name or as an argument to chdir.

21 EISDIR Is a directory
An attempt to write on a directory.

22 EINVAL Invalid argument
Some invalid argument: dismounting a non-mounted device, mentioning an unknown signal in signal, reading or writing a file for which seek has generated a negative pointer. 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 opens can be accepted.

24 EMFILE Too many open files
Customary configuration limit is 20 per process.

25 ENOTTY Not a typewriter
The file mentioned in stty or gtty is not a terminal or one of the other devices to which these calls apply.

26 ETXTBSY Text file busy
An attempt to execute a pure-procedure program that is currently open for writing (or reading!). Also an attempt to open for writing a pure-procedure program that is being executed.

27 EFBIG File too large
The size of a file exceeded the maximum (about 1.0E9 bytes).

28 ENOSPC No space left on device
During a write to an ordinary file, there is no free space left on the device.

29 ESPIPE Illegal seek
An lseek was issued to a pipe. This error should 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 device mounted read-only.

31 EMLINK Too many links
An attempt to make more than 32767 links to a file.

32 EPIPE Broken pipe
A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

33 EDOM Math argument
The argument of a function in the math package (3M) is out of the domain of the function.

34 ERANGE Result too large
The value of a function in the math package (3M) is unrepresentable within machine precision.

35 ETPL Fatal error - tape position lost
A magtape driver has encountered a tape error that has caused it to loose track of tape position.

36 ETOL Tape unit off-line
An attempt has been made to access a tape drive that is not available, i.e., no tape loaded or off-line.

37 ETWL Tape unit write locked
A write operation was attempted on a tape without a write enable ring installed.

38 ETO Tape unit already open
Caused by an attempt to access a tape drive that is already in use or is hung for some reason.

SEE ALSO
intro(3)

RESTRICTIONS
The getfp(), bdflush(), errlog(), and ttlocl() system calls are intentionally not documented. These calls are reserved for use by ULTRIX-11 system programs and must not be called by any user process.

ASSEMBLER

as /usr/include/sys.s file ...

The PDP11 assembly language interface is given for each system call. The assembler symbols are defined in '/usr/include/sys.s'.

Return values appear in registers r0 and r1; it is unwise to count on these registers being preserved when no value is expected. An erroneous call is always indicated by turning on the c-bit of the condition codes. The error number is returned in r0. The presence of an error is most easily tested by the instructions bes and bec ('branch on error set (or clear)'). These are synonyms for the bcs and bcc instructions.

On the Interdata 8/32, the system call arguments correspond well to the arguments of the C routines. The sequence is:

```
la   %2,errno
l    %0,&callno
svc  0,args
```

Thus register 2 points to a word into which the error number will be stored as needed; it is cleared if no error occurs. Register 0 contains the system call number; the nomenclature is identical to that on the PDP11. The argument of the svc is the address of the arguments, laid out in storage as in the C calling sequence. The return value is in register 2 (possibly 3 also, as in pipe) and is -1 in case of error. The overflow bit in the program status word is also set when errors occur.

ACCESS(2)

NAME

access - determine accessibility of file

SYNOPSIS

```
access(name, mode)
char *name;
```

DESCRIPTION

Access checks the given file name for accessibility according to mode, which is 4 (read), 2 (write) or 1 (execute) or a combination thereof. Specifying mode 0 tests whether the directories leading to the file can be searched and the file exists.

An appropriate error indication is returned if name cannot be found or if any of the desired access modes would not be granted. On disallowed accesses -1 is returned and the error code is in errno. 0 is returned from successful tests.

The user and group IDs with respect to which permission is checked are the real UID and GID of the process, so this call is useful to set-UID programs.

Notice that it is only access bits that are checked. A directory may be announced 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 exec will fail unless it is in proper format.

DIAGNOSTICS

Access to the file is denied if:

[EACCES] Permission bits of the file mode do not permit the requested access.

[EACCES] Search permission is denied on a component of the path prefix. 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.

[EROFS] Write access is requested for a file on a read-only file system.

[ETXTBSY] Write access is requested for a pure procedure (shared text) file that is being executed.

SEE ALSO
stat(2)

ASSEMBLER
(access = 33.)
sys access; name; mode

ACCT(2)

NAME

acct - turn accounting on or off

SYNOPSIS

```
acct(file)
char *file;
```

DESCRIPTION

The system is prepared to write a record in an accounting file for each process as it terminates. This call, with a null-terminated string naming an existing file as argument, turns on accounting; records for each terminating process are appended to file. An argument of 0 causes accounting to be turned off.

The accounting file format is given in acct(5).

On error -1 is returned. The file must exist and the call may be exercised only by the superuser. It is erroneous to try to turn on accounting when it is already on.

DIAGNOSTICS

Acct will fail if one of the following is true:

[EACCES]	File is not a regular file.
[EBUSY]	An attempt is made to turn on accounting when it is already on.
[EFAULT]	File points outside the process's allocated address space.
[ENOENT]	The named file does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.
[EPERM]	The caller is not the superuser.
[EROFS]	The file resides on a read-only file system.

RESTRICTIONS

No accounting is produced for programs running when a crash occurs. In particular nonterminating programs are never accounted for.

SEE ALSO

acct(5), sa(1)

ASSEMBLER

```
(acct = 51.)
sys acct; file
```

NAME

alarm - schedule signal after specified time

SYNOPSIS

```
alarm(seconds)
unsigned seconds;
```

DESCRIPTION

Alarm causes signal SIGALRM, see signal(2), to be sent 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 cancelled. Because the clock has a 1-second resolution, the signal may occur up to one second early; 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 65535 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

pause(2), signal(2), sleep(3)

ASSEMBLER

```
(alarm = 27.)
(seconds in r0)
sys alarm
(previous amount in r0)
```

BRK(2)

NAME

brk, sbrk, break - change core allocation

SYNOPSIS

```
char *brk(addr)

char *sbrk(incr)
```

DESCRIPTION

Brk sets the system's idea of the lowest location not used by the program (called the break) to addr (rounded up to the next multiple of 64 bytes on the PDP11, 256 bytes on the Interdata 8/32, 512 bytes on the VAX-11/780). Locations not less than addr and below the stack pointer are not in the address space and will thus cause a memory violation if accessed.

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 via exec the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use break.

RETURN VALUE

Zero is returned if the break could be set; -1 if the program requests more memory than the system limit or if too many segmentation registers would be required to implement the break.

DIAGNOSTICS

Sbrk will fail and no additional memory will be allocated if:

[ENOMEM] The maximum possible size of a data segment (compiled into the system) was exceeded.

[ENOMEM] The maximum available memory for a user process was exceeded. For I/D process, text or data+stack was larger than 64KB. For non-I/D process, text+data+stack was larger than 64KB.

RESTRICTIONS

Setting the break in the range 0177701 to 0177777 (on the PDP11) is the same as setting it to zero.

SEE ALSO

exec(2), malloc(3), end(3)

DUP(2)

NAME

dup, dup2 - duplicate an open file descriptor

SYNOPSIS

```
dup(fildes)
int fildes;

dup2(fildes, fildes2)
int fildes, fildes2;
```

DESCRIPTION

Given a file descriptor returned from an open, pipe, or creat call, dup allocates another file descriptor synonymous with the original. The new file descriptor is returned.

In the second form of the call, fildes is a file descriptor referring to an open file, and fildes2 is a non-negative integer less than the maximum value allowed for file descriptors (approximately 19). Dup2 causes fildes2 to refer to the same file as fildes. If fildes2 already referred to an open file, it is closed first.

DIAGNOSTICS

Dup and dup2 will fail if:

[EBADF] Fildes or fildes2 is not a valid active descriptor.

[EMFILE] Too many descriptors are active.

SEE ALSO

creat(2), open(2), close(2), pipe(2)

ASSEMBLER

```
(dup = 41.)
(file descriptor in r0)
(new file descriptor in r1)
sys dup
(file descriptor in r0)
```

The dup2 entry is implemented by adding 0100 to fildes.

ASSEMBLER

```
(break = 17.)
sys break; addr
```

Break performs the function of brk. The name of the routine differs from that in C for historical reasons.

CHDIR(2)

NAME

chdir, chroot - change default directory

SYNOPSIS

```
chdir(dirname)
char *dirname;

chroot(dirname)
char *dirname;
```

DESCRIPTION

Dirname is the address of the pathname of a directory, terminated by a null byte. Chdir causes this directory to become the current working directory, the starting point for path names not beginning with '/'.

Chroot sets the root directory, the starting point for path names beginning with '/'. The call is restricted to the superuser.

RETURN VALUE

Zero is returned if the directory is changed; -1 is returned if the given name is not that of a directory or is not searchable.

DIAGNOSTICS

Chdir will fail and the current working directory will be unchanged if one or more of the following is true:

[EACCES] Search permission is denied for a component of the path name.

[EFAULT] Dirname points outside the process's allocated address space.

[ENFILE] Insufficient system space to contain i-node.

[ENOENT] The named directory, or an element within the named path, does not exist.

[ENOTDIR] A component of the path name is not a directory.

Chroot will also fail if:

[EPERM] The user is not the superuser.

SEE ALSO

cd(1)

ASSEMBLER

(chdir = 12.)

```
sys chdir; dirname

(chroot = 61.)
sys chroot; dirname
```

CHMOD(2)

NAME

chmod - change mode of file

SYNOPSIS

```
chmod(name, mode)
char *name;
```

DESCRIPTION

The file whose name is given as the null-terminated string pointed to by name has its mode changed to mode. Modes are constructed by ORing together some combination of the following:

```
04000 set user ID on execution
02000 set group ID on execution
01000 save text image after execution
00400 read by owner
00200 write by owner
00100 execute (search on directory) by owner
00070 read, write, execute (search) by group
00007 read, write, execute (search) by others
```

If an executable file is set up for sharing (-n or -i option of ld(1)) then mode 1000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time. Ability to set this bit is restricted to the superuser since swap space is consumed by the images; it is only worth while for heavily used commands.

Only the owner of a file (or the superuser) may change the mode. Only the superuser can set the 1000 mode.

RETURN VALUE

Zero is returned if the mode is changed; -1 is returned if name cannot be found or if current user is neither the owner of the file nor the superuser.

DIAGNOSTICS

Chmod will fail and the file mode will be unchanged if:

[EACCES] Search permission is denied on a component of the path prefix.

[EFAULT] Name points outside the process's allocated address space.

[ENFILE] Insufficient system space to contain i-node.

[ENOENT] The named file, or an element within the

named file, does not exist.

[ENOTDIR] A component of the path prefix is not a directory.

[EPERM] The effective user ID does not match the owner of the file and the effective user ID is not the superuser.

[EROFS] The named file resides on a read-only file system.

SEE ALSO
chmod(1)

ASSEMBLER
(chmod = 15.)
sys chmod; name; mode

NAME

chown - change owner and group of a file

SYNOPSIS

```
chown(name, owner, group)
char *name;
```

DESCRIPTION

The file whose name is given by the null-terminated string pointed to by name has its owner and group changed as specified. Only the superuser may execute this call, because if users were able to give files away, they could defeat the (nonexistent) file-space accounting procedures.

RETURN VALUE

Zero is returned if the owner is changed; -1 is returned on illegal owner changes.

DIAGNOSTICS

Chown will fail and the file will be unchanged if:

[EACCES]	Search permission is denied on a component of the path prefix.
[EFAULT]	Name points outside the process's allocated address space.
[ENFILE]	Insufficient system space to contain i-node.
[ENOENT]	The named file, or an element within path name, does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.
[EPERM]	The effective user ID does not match the owner of the file and the effective user ID is not the superuser.
[EROFS]	The named file resides on a read-only file system.

SEE ALSO

chown(1), passwd(5)

ASSEMBLER

```
(chown = 16.)
sys chown; name; owner; group
```

NAME

close - close a file

SYNOPSIS

close(fildes)

DESCRIPTION

Given a file descriptor such as returned from an open, creat, dup or pipe(2) call, close closes the associated file. A close of all files is automatic on exit, but since there is a limit on the number of open files per process, close is necessary for programs which deal with many files.

Files are closed upon termination of a process, and certain file descriptors may be closed by exec(2) (see ioctl(2)).

RETURN VALUE

Zero is returned if a file is closed; -1 is returned for an unknown file descriptor.

DIAGNOSTICS

Close will fail if:

[EBADF] Fildes is not an active descriptor.

SEE ALSO

creat(2), open(2), pipe(2), exec(2), ioctl(2)

ASSEMBLER

(close = 6.)
(file descriptor in r0)
sys close

NAME

creat - create a new file

SYNOPSIS

```
creat(name, mode)
char *name;
```

DESCRIPTION

Creat creates a new file or prepares to rewrite an existing file called name, given as the address of a null-terminated string. 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 did exist, its mode and owner remain unchanged but it is truncated to 0 length.

The file is also opened for writing, and its file descriptor is returned.

The mode given is arbitrary; it need not allow writing. This feature is used by programs which deal with temporary files of fixed names. The creation is done with a mode that forbids writing. Then if a second instance of the program attempts a creat, an error is returned and the program knows that the name is unusable for the moment.

RETURN VALUE

Zero is returned if the file is created; -1 is returned otherwise.

DIAGNOSTICS

Creat will fail and the file will not be created if:

[EACCES] A component of the path prefix denies search permission.

[EACCES] The directory in which the file is to be created is not writable by the user.

[EFAULT] Name points outside the process's allocated address space.

[EISDIR] The named file is a directory.

[EMFILE] The maximum number of file descriptors allowed are already open.

[ENFILE] No more system file descriptors are available, or there is insufficient system space to contain the i-node.

[ENOENT] Element within path name does not exist.

[ENOTDIR] A component of the path prefix is not a directory.

[ENXIO] Device special file within name is not for the current system (major device number is greater than "nchrdev").

[EROFS] The directory in which the file is to be created is on a read-only file system.

[ETXTBSY] The file is a pure procedure (shared text) file that is being executed.

SEE ALSO

write(2), close(2), chmod(2), umask (2)

ASSEMBLER

(creat = 8.)
sys creat; name; mode
(file descriptor in r0)

NAME

ac - login accounting

SYNOPSIS

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. -w is used to specify an alternate wtmp file. -p prints individual totals; without this option, only totals are printed. -d causes a 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, /usr/adm/wtmp is used.

The accounting file /usr/adm/wtmp is maintained by init and login. 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.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), login(1), utmp(5).

NAME

execl, execv, execle, execve, execlp, execvp, exec, exece, environ - execute a file

SYNOPSIS

```
execl(name, arg0, argl, ..., argn, 0)
char *name, *arg0, *argl, ..., *argn;

execv(name, argv)
char *name, *argv[ ];

execle(name, arg0, argl, ..., argn, 0, envp)
char *name, *arg0, *argl, ..., *argn, *envp[ ];

execve(name, argv, envp);
char *name, *argv[ ], *envp[ ];

extern char **environ;
```

DESCRIPTION

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.

Files remain open across exec unless explicit arrangement has been made; see ioctl(2). Ignored signals remain ignored across these calls, but signals that are caught (see signal(2)) are reset to their default values.

Each user 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 his access privileges. Exec changes the effective user and group ID to the owner of the executed file if the file has the 'set-user-ID' or 'set-group-ID' modes. The real user ID is not affected.

The name 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.

From C, 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 0 argument must end the argument list.

The execv version is useful when the number of arguments is unknown in advance; the arguments to execv are the name of

EXEC(2)

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(5) 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. The exec routines use lower-level routines as follows to pass an environment explicitly:

```
execle(file, arg0, arg1, . . . , argn, 0, environ);
execve(file, argv, environ);
```

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

/bin/sh shell, invoked if command file found by execlp or execvp

RETURN VALUE

If exec 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.

DIAGNOSTICS

Exec will fail and return to the calling process if one or more of the following is true:

[EACCES]	Search permission is denied for a directory listed in the new process file's path prefix.
[EACCES]	The new process file mode denies execute permission.
[EACCES]	The new process file is not a regular file.
[EFAULT]	Name points outside the process's allocated address space.
[EFAULT]	Name, argv, or envp point to an illegal address (outside of the user's image).
[ENFILE]	Insufficient system space to contain i-node.
[ENOENT]	One or more components of the new process file's path name does not exist.
[ENOENT]	The target file is a directory.
[ENOEXEC]	The new process file has the appropriate access permission but the file header contains an invalid magic number.
[ENOEXEC]	The text or data segment is null.
[ENOEXEC]	An overlay is larger than the overlay max size.
[ENOEXEC]	Unable to read file header.
[ENOEXEC]	Found a bad magic number in the file header.
[ENOMEM]	The new process requires more memory than is allowed.
[ENOTDIR]	A component of the new process path name is not a directory.
[ETXTBSY]	The new process file is a pure procedure (shared text) file that is currently open for reading or writing by some process.

RESTRICTIONS

If execvp is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of argv[0] and argv[-1] will be modified before return.

SEE ALSO

fork(2), environ(5)

EXEC(2)

ASSEMBLER

```
(exec = 11.)
sys exec; name; argv

(exece = 59.)
sys exece; name; argv; envp
```

Plain exec is obsoleted by exece, but remains for historical reasons.

When the called file starts execution on the PDP11, the stack pointer points to a word containing the number of arguments. Just above this number is a list of pointers to the argument strings, followed by a null pointer, followed by the pointers to the environment strings and then another null pointer. The strings themselves follow; a 0 word is left at the very top of memory.

```
  sp->    nargs
     arg0
     ...
     argn
     0
     env0
     ...
     envm
     0

 arg0:    <arg0\0>
     ...
 env0:    <env0\0>
     0
```

On the Interdata 8/32, the stack begins at a conventional place (currently 0xD0000) and grows upwards. After exec, the layout of data on the stack is as follows.

```
     int  0
 arg0:    byte ...
     ...
argp0:    int  arg0
     ...
     int  0
envp0:    int  env0
     ...
     int  0
 %2->     space     40
     int  nargs
     int  argp0
     int  envp0
 %3->
```

EXEC(2)

This arrangement happens to conform well to C calling conventions.

EXIT(2)

NAME

exit - terminate process

SYNOPSIS

```
exit(status)
int status;

_exit(status)
int status;
```

DESCRIPTION

Exit is the normal means of terminating a process. Exit closes all the process's files and notifies the parent process if it is executing a wait. The low-order 8 bits of status are available to the parent process.

This call can never return.

The C function exit may cause cleanup actions before the final 'sys exit'. The function _exit circumvents all cleanup.

SEE ALSO

wait(2)

ASSEMBLER

```
(exit = 1.)
(status in r0)
sys exit
```

NAME

fork - spawn new process

SYNOPSIS

fork( )

DESCRIPTION

Fork is the only way new processes are created. The new process's core image is a copy of that of the caller of fork. The only distinction is the fact that the value returned in the old (parent) process contains the process ID of the new (child) process, while the value returned in the child is 0. Process ID's range from 1 to 30,000. This process ID is used by wait(2).

Files open before the fork are shared, and have a common read-write pointer. In particular, this is the way that standard input and output files are passed and also how pipes are set up.

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.

Only the superuser can take the last process-table slot.

DIAGNOSTICS

Fork will fail and no child process will be created if:

[EAGAIN] The system-imposed limit on the total number of processes under execution would be exceeded.

[EAGAIN] The system-imposed limit on the total number of processes under execution by a single user would be exceeded.

[ENOMEM] There is no swap space available for "maxmem".

SEE ALSO

wait(2), exec(2)

ASSEMBLER

(fork = 2.)
sys fork
(new process return)
(old process return, new process ID in r0)

FORK(2)

The return locations in the old and new process differ by one word. The C-bit is set in the old process if a new process could not be created.

NAME

fperr - get floating point error status

SYNOPSIS

```
fperr(&fpstat)
int fpstat[3];      /* FP status register */
                    /* FP error code */
                    /* FP error address */
```

DESCRIPTION

The fperr system call loads the contents of the floating point hardware status and error registers into fpstat. This call allows user processes to handle floating point exceptions.

FPSIM(2)

NAME

fpsim - report or change the status of floating point simulation

SYNOPSIS

fpsim(flag)

DESCRIPTION

Fpsim is used to enable, disable, or get the status of the kernel floating point simulator. If flag is 2, the current status of the simulator is returned. The return value will be either 0, 1, or 2, which respectively means that the simulator is disabled, enabled, or not configured into the current system.

If flag is 0 or 1 the simulator is respectively disabled or enabled, and the old status of the simulator is returned. Only the super user is allowed to change the status of the simulator.

ERRORS

fpsim will fail and the status of the simulator will remain unchanged if:

[EPERM]	The effective user ID of the calling process is not the superuser.
[ENODEV]	The simulator is not configured into the current system.
[EINVAL]	A mode other than 0, 1 or 2 was specified.

SEE ALSO

fpsim(1)

RESTRICTIONS

This call is only temporary, it is intended only for use if the kernel floating point simulator should blow up. Once it has been more thoroughly tested and verified there will no longer be a need to be able to turn it off on a running system.

NAME

setpgrp, getpgrp - set/get process group

SYNOPSIS

```
int getpgrp(pid)

setpgrp(pid, pgrp)

cc ... -ljobs
```

DESCRIPTION

The process group of the specified process is returned by getpgrp. Setpgrp sets the process group of the specified process pid to the specified pgrp. If pid is zero, then the call applies to the current process.

If the invoker is not the superuser, then the affected process must have the same effective user-id as the invoker or be a descendant of the invoking process.

This call is used by csh(1) to create process groups in implementing job control. The TIOCGPGRP and TIOCSPGRP calls described in tty(4) are used to get/set the process group of the control terminal.

See intro(3J) for a general discussion of job control.

DIAGNOSTICS

Setpgrp will fail and the process group will not be altered if one of the following is true:

[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.

[ESRCH] The requested process does not exist.

RESTRICTIONS

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of these system calls and even the calls themselves are thus subject to change.

A system call setpgrp has been implemented in other versions of UNIX which are not widely used outside of Bell Laboratories; these implementations have, in general, slightly different semantics.

SEE ALSO
csh(1), getuid(2), intro(3J), tty(4)

ASSEMBLER
(setpgrp = 39.)
(process id in r0)
sys setpgrp; newgrp

(getpgrp is implemented as setpgrp(pid,-1))
sys setgrp; -1
(process group in r0)

NAME

getpid, getppid - get process identification

SYNOPSIS

```
getpid( )
getppid( )
```

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

mktemp(3)

ASSEMBLER

```
(getpid = 20.)
sys getpid
(pid in r0)
(ppid in r1)
```

NAME

getuid, getgid, geteuid, getegid - get user and group identity

SYNOPSIS

```
getuid( )

geteuid( )

getgid( )

getegid( )
```

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, in contradistinction to the effective user ID, which determines his access permission at the moment. It is thus useful to programs which operate using the 'set user ID' mode, to find out who invoked them.

Getgid returns the real group ID, getegid the effective group ID.

SEE ALSO

setuid(2)

ASSEMBLER

```
(getuid = 24.)
sys getuid
(real user ID in r0, effective user ID in r1)

(getgid = 47.)
sys getgid
(real group ID in r0, effective group ID in r1)
```

NAME

ghostname, shostname - get/set name of current host

SYNOPSIS

```
ghostname(name, namelen)
char *name;
int namelen;

shostname(name, namelen)
char *name;
int namelen;
```

DESCRIPTION

Ghostname returns the standard host name for the current processor, as previously set by shostname. The parameter namelen specifies the size of the name array. The returned name is null-terminated unless insufficient space is provided.

Shostname sets the name of the host machine to be name, which has length namelen. This call is restricted to the superuser 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 int 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 superuser.

RESTRICTIONS

Host names are limited to 32 characters.

These system calls are the same as the gethostname and sethostname system calls in ULTRIX-32. Due to the current constraint that function names must be significant in the first seven characters, the names were shortend to keep them from clashing with the (not yet implemented) gethostid and sethostid system calls.

For compatibility reasons, you can use gethostname and sethostname in your program, provided that either the top of the program contains the line

```
#define gethostname ghostname
```

or the module is compiled with
-Dgethostname=ghostname

as an argument to cc. This works because cpp recognizes eight characters of significance.

NAME

indir - indirect system call

ASSEMBLER

```
(indir = 0.)
sys indir; call
```

The system call at the location call is executed. Execution resumes after the indir call.

The main purpose of indir is to allow a program to store arguments in system calls and execute them out of line in the data segment. This preserves the purity of the text segment.

If indir is executed indirectly, it is a no-op. If the instruction at the indirect location is not a system call, indir returns error code EINVAL; see intro(2).

IOCTL(2)

NAME

ioctl, stty, gtty - control device

SYNOPSIS

```
#include <sgtty.h>

ioctl(fildes, request, argp)
struct sgttyb *argp;

stty(fildes, argp)
struct sgttyb *argp;

gtty(fildes, argp)
struct sgttyb *argp;
```

DESCRIPTION

Ioctl performs a variety of functions on character special files (devices). The writeups of various devices in section 4 discuss how ioctl applies to them.

For certain status setting and status inquiries about terminal devices, the functions stty and gtty are equivalent to

```
ioctl(fildes, TIOCSETP, argp)
ioctl(fildes, TIOCGETP, argp)
```

respectively; see tty(4).

The following two calls, however, apply to any open file:

```
ioctl(fildes, FIOCLEX, NULL);
ioctl(fildes, FIONCLEX, NULL);
```

The first causes the file to be closed automatically during a successful exec operation; the second reverses the effect of the first.

RETURN VALUE

Zero is returned if the call was successful; -1 if the file descriptor does not refer to the kind of file for which it was intended.

DIAGNOSTICS

Ioctl will fail if one or more of the following is true:

[EBADF] Fildes is not a valid descriptor.

[EBADF] Carrier was lost on the terminal line.

[EFAULT] Argp points to an illegal address.

[ENOTTY] The specified request does not apply to the kind of object that the descriptor fildes

references.

[ENXIO] An attempt was made to set an invalid line discipline.

[EPERM] The specified ioctl() function is only valid for the superuser (TIOCLOCAL, TIOCSMLB, TIOCCMLB).

RESTRICTIONS

Strictly speaking, since ioctl may be extended in different ways to devices with different properties, argp should have an open-ended declaration like

```
union { struct sgttyb ...; ... } *argp;
```

The important thing is that the size is fixed by 'struct sgttyb'.

SEE ALSO

stty(1), tty(4), exec(2)

ASSEMBLER

```
(ioctl = 54.)
sys ioctl; fildes; request; argp

(stty = 31.)
(file descriptor in r0)
stty; argp

(gtty = 32.)
(file descriptor in r0)
sys gtty; argp
```

KILL(2)

NAME

kill - send signal to a process

SYNOPSIS

```
kill(pid, sig);
```

DESCRIPTION

Kill sends the signal sig to the process specified by the process number in r0. See signal(2) for a list of signals.

The sending and receiving processes must have the same effective user ID, otherwise this call is restricted to the superuser.

If the process number is 0, the signal is sent to all other processes in the sender's process group; see tty(4).

If the process number is -1, and the user is the superuser, the signal is broadcast universally except to processes 0 and 1, the scheduler and initialization processes, see init(8).

Processes may send signals to themselves.

RETURN VALUE

Zero is returned if the process is killed; -1 is returned otherwise, with errno containing the error code.

DIAGNOSTICS

Kill will fail and no signal will be sent if any of the following occur:

[EINVAL]	Sig is not a valid signal number.
[EINVAL]	Attempt was made to send to an unassigned process group.
[ESRCH]	No process can be found corresponding to that specified by pid.
[ESRCH]	No processes killed.

SEE ALSO

signal(2), kill(1)

ASSEMBLER

```
(kill = 37.)
(process number in r0)
sys kill; sig
```

NAME

killpg - send signal to a process or a process group

SYNOPSIS

killpg(pgrp, sig)

cc ... -ljobs

DESCRIPTION

Killpg sends the signal sig to the specified process group. See sigsys(2) for a list of signals; see intro(3J) for an explanation of process groups.

The sending process and members of the process group must have the same effective user ID, otherwise this call is restricted to the superuser. As a single special case the continue signal SIGCONT may be sent to any process which is a descendant of the current process. This allows a command interpreter such as csh(1) to restart set-user-id processes stopped from the keyboard with a stop signal.

The calls

killpg(0, sig)

and

kill(0, sig)

have identical effects, sending the signal to all members of the invoker's process group (including the process itself). It is preferable to use the call involving kill in this case, as it is portable to other UNIX systems.

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.

DIAGNOSTICS

Killpg will fail and no signal will be sent if any of the following occur:

[EINVAL]	Sig is not a valid signal number.
[EINVAL]	Attempt to send to an unassigned process group.
[EPERM]	The caller is not the superuser.
[ESRCH]	No process can be found corresponding to the process group specified by pgrp.

KILLPG(2J)

[ESRCH] No processes killed.

RESTRICTIONS

The job control facilities are not available in standard Version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of this system call and even the call itself are thus subject to change.

SEE ALSO

csh(1), kill(1), kill(2), signal(2), sigsys(2J), intro(3J)

ASSEMBLER (PDP-11)

```
(kill =37.)
(process number in r0)
sys kill; -sig (negative signal number means killpg)
```

NAME

link - link to a file

SYNOPSIS

```
link(name1, name2)
char *name1, *name2;
```

DESCRIPTION

A link to name1 is created; the link has the name name2. Either name may be an arbitrary path name.

DIAGNOSTICS

Link will fail and no link will be created if:

[EACCES]	A component of either path prefix denies search permission.
[EACCES]	The requested link requires writing in a directory with a mode that denies write permission.
[EEXIST]	A file by the same name already exists.
[EFAULT]	One of the path names specified is outside the process's allocated address space.
[ENFILE]	Insufficient system space to contain i-node.
[ENOTDIR]	A path prefix component is not a directory.
[ENOENT]	A path prefix component does not exist.
[ENOENT]	The file named by name1 does not exist.
[EPERM]	The file named by name1 is a directory and the effective user ID is not the superuser.
[EROFS]	The requested link requires writing in a directory on a read-only file system.
[EXDEV]	The link named by name2 and the file named by name1 are on different file systems.
[E2BIG]	The argument count exceeded "ncargs".

SEE ALSO

ln(1), unlink(2)

ASSEMBLER

```
(link = 9.)
sys link; name1; name2
```

LOCK(2)

NAME

lock - lock a process in primary memory

SYNOPSIS

lock(flag)

DESCRIPTION

If the flag argument is non-zero, the process executing this call will not be swapped except if it is required to grow. If the argument is zero, the process is unlocked. This call may only be executed by the superuser.

DIAGNOSTICS

Lock will fail if:

[EPERM] The effective user ID is not the superuser.

RESTRICTIONS

Locked processes interfere with the compaction of primary memory and can cause deadlock. This system call is not considered a permanent part of the system.

ASSEMBLER

```
(lock = 53.)
sys lock; flag
```

NAME

lseek, tell - move read/write pointer

SYNOPSIS

```
long lseek(fildes, offset, whence)
long offset;

long tell(fildes)
```

DESCRIPTION

The file descriptor refers to a file open for reading or writing. The read (resp. write) pointer for the file is set as follows:

If whence is 0, the pointer is set to offset bytes.

If whence is 1, the pointer is set to its current location plus offset.

If whence is 2, the pointer is set to the size of the file plus offset.

The returned value is the resulting pointer location.

The obsolete function tell(fildes) is identical to lseek(fildes, 0L, 1).

Seeking far beyond the end of a file, then writing, creates a gap or 'hole', which occupies no physical space and reads as zeros.

RETURN VALUE

Upon successful completion, the new file offset is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

DIAGNOSTICS

Lseek will fail and the file pointer will remain unchanged if:

[EBADF] Fildes is not an open file descriptor.

[ESPIPE] Fildes is associated with a pipe.

RESTRICTIONS

Lseek is a no-op on character special files.

SEE ALSO

open(2), creat(2), fseek(3)

ASSEMBLER

(lseek = 19.)

LSEEK(2)

(file descriptor in r0)
sys lseek; offset1; offset2; whence

Offset1 and offset2 are the high and low words of offset; r0 and r1 contain the pointer upon return.

NAME

mknod - make a directory or a special file

SYNOPSIS

```
mknod(name, mode, addr)
char *name;
```

DESCRIPTION

Mknod creates a new file whose name is the null-terminated string pointed to by name. The mode of the new file (including directory and special file bits) is initialized from mode. (The protection part of the mode is modified by the process's mode mask; see umask(2)). The first block pointer of the i-node is initialized from addr. For ordinary files and directories addr is normally zero. In the case of a special file, addr specifies which special file.

Mknod may be invoked only by the superuser.

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.

DIAGNOSTICS

Mknod will fail and the file mode will be unchanged if:

[EACCES]	Access to a directory file denied.
[EACCES]	A component of the path prefix denies search permission.
[EEXIST]	The named file exists.
[EFAULT]	Name points outside the process's allocated address space.
[ENOENT]	A component of the path prefix does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.
[ENOSPC]	No i-nodes are available on the device.
[EPERM]	The effective user ID does not match the owner of the file and the effective user ID is not the superuser.
[EROFS]	The named file resides on a read-only file system.

MKNOD(2)

SEE ALSO
mkdir(1), mknod(1), filsys(5)

ASSEMBLER
(mknod = 14.)
sys mknod; name; mode; addr

NAME

mount, umount - mount or remove file system

SYNOPSIS

```
mount(special, name, rwflag)
char *special, *name;

umount(special)
char *special;
```

DESCRIPTION

Mount announces to the system that a removable file system has been mounted on the block-structured special file special; from now on, references to file name will refer to the root file on the newly mounted file system. Special and name are pointers to null-terminated strings containing the appropriate path names.

Name must exist already. Name must be a directory (unless the root of the mounted file system is not a directory). Its old contents are inaccessible while the file system is mounted.

The rwflag argument determines whether the file system can be written on; if it is 0 writing is allowed, if non-zero no writing is done. 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.

Umount announces to the system that the special file is no longer to contain a removable file system. The associated file reverts to its ordinary interpretation.

DIAGNOSTICS

Mount will fail if:

[EACCES]	A component of the path prefix denies search permission.
[EBUSY]	Name is not a directory or another process currently holds a reference to it.
[EBUSY]	No space remains in the mount table.
[ENODEV]	Special does not exist.
[ENOENT]	An element within name does not exist.
[ENOTBLK]	Special is not for a block oriented device.
[ENOTDIR]	A component of the path prefix in special or

name is not a directory.

[ENXIO] The major device number of special is out of range (this indicates no device driver exists for the associated hardware).

[EPERM] The process's effective user ID is not the superuser.

[EROFS] Name resides on a read-only file system.

Umount may fail with one of the following errors:

[EBUSY] A process is holding a reference to a file located on the file system.

[EINVAL] The requested device is not in the mount table.

[ENODEV] Special does not exist.

[ENOTBLK] Special is not a block device.

[ENXIO] The major device number of special is out of range (this indicates no device driver exists for the associated hardware).

[EPERM] The process's effective user ID is not the superuser.

SEE ALSO
mount(1)

ASSEMBLER
```
(mount = 21.)
sys mount; special; name; rwflag

(umount = 22.)
sys umount; special
```

NAME

nice - set program priority

SYNOPSIS

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 flak from the administration.

Negative increments are ignored except on behalf of the superuser. The priority is limited to the range -20 (most urgent) to 20 (least).

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).

DIAGNOSTICS

Nice will fail and the process's scheduling priority remain the same if:

[EPERM] Incr is negative and the process's effective user ID is not the superuser.

SEE ALSO

nice(1)

ASSEMBLER

```
(nice = 34.)
(priority in r0)
sys nice
```

NOSTK(2)

NAME

nostk - allow process to manage its own stack

SYNOPSIS

nostk()

DESCRIPTION

Nostk informs the system that the process wishes to manage its own stack. The system releases the stack segment(s) it has reserved, making them available for allocation (via brk(2)) by the user.

C program should use nostk only with great caution and understanding of the C language calling and stack conventions. It is most useful for assembler programs that want to use the entier available address space.

ASSEMBLER

(nostk = 84.)
sys nostk

NAME

open - open for reading or writing

SYNOPSIS

```
open(name, mode)
char *name;
```

DESCRIPTION

Open opens the file name for reading (if mode is 0), writing (if mode is 1) or for both reading and writing (if mode is 2). Name is the address of a string of ASCII characters representing a path name, terminated by a null character.

The file is positioned at the beginning (byte 0). The returned file descriptor must be used for subsequent calls for other input-output functions on the file.

DIAGNOSTICS

The named file is opened unless one or more of the following is true:

[EACCES]	A component of the path prefix denies search permission.
[EBUSY]	The device specified is already open for exclusive use and the caller is not the superuser.
[EFAULT]	Name points outside the process's allocated address space.
[EISDIR]	The named file is a directory and the arguments specify it is to be opened for writing.
[EMFILE]	The maximum number of file descriptors allowed are already open.
[ENFILE]	No more system file descriptors are available.
[ENFILE]	Insufficient system space to contain i-node.
[ENOENT]	The named file does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.
[ENXIO]	Device special file is not for the current system (major device number is greater than "nchrdev", or greater than "nblkdev").
[ENXIO]	An attempt was made to open /dev/tty without

a controlling terminal.

[ENXIO] An illegal device was specified, or the device is already open.

[EROFS] The named file resides on a read-only file system and the file is to be modified.

[ETO] The specified tape drive is already open.

[ETOL] The specified tape drive is not on-line.

[ETPL] Tape position was lost for the specified tape drive.

[ETWL] The specified tape drive is physically write-locked.

[ETXTBSY] The file is a pure procedure (shared text) file that is being executed and the open call has requested write access.

SEE ALSO

creat(2), read(2), write(2), dup(2), close(2)

ASSEMBLER

```
(open = 5.)
sys open; name; mode
(file descriptor in r0)
```

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(2) or alarm(2).

SEE ALSO
kill(1), kill(2), alarm(2), signal(2), setjmp(3)

ASSEMBLER
(pause = 29.)
sys pause

PHYS(2)

NAME

phys - allow a process to access physical addresses

SYNOPSIS

phys(segreg, size, physadr)

DESCRIPTION

The argument segreg specifies a process virtual (data-space) address range of 8K bytes starting at virtual address segregx8K bytes. This address range is mapped into physical address physadrx64 bytes. Only the first sizex64 bytes of this mapping is addressable. If size is zero, any previous mapping of this virtual address range is nullified. For example, the call

```
phys(6, 1, 0177775);
```

will map virtual addresses 0160000-0160077 into physical addresses 017777500-017777577. In particular, virtual address 0160060 is the PDP-11 console located at physical address 017777560.

This call may only be executed by the superuser.

DIAGNOSTICS

Phys will fail and the segmentation registers be unaffected if:

[EINVAL] Segreg is less than 0 or greater than 7.

[EINVAL] Size is less than 0 or greater than 128.

[EINVAL] An attempt was made to map a segment which was already mapped by some method other than by phys.

[EPERM] The process's effective user ID is not the superuser.

RESTRICTIONS

This system call is obviously very machine dependent and very dangerous. This system call is not considered a permanent part of the system.

SEE ALSO

PDP-11 segmentation hardware

ASSEMBLER

```
(phys = 52.)
sys phys; segreg; size; physadr
```

NAME

pipe - create an interprocess channel

SYNOPSIS

```
pipe(fildes)
int fildes[2];
```

DESCRIPTION

The pipe system call creates an I/O mechanism called a pipe. The file descriptors returned can be used in read and write operations. When the pipe is written using the descriptor fildes[1] up to 4096 bytes of data are buffered before the writing process is suspended. A read using the descriptor fildes[0] will pick up the data. Writes with a count of 4096 bytes or less are atomic; no other process can intersperse data.

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. A signal is generated if a write on a pipe with only one end is attempted.

DIAGNOSTICS

The pipe call will fail if:

[EFAULT]	The fildes buffer is in an invalid area of the process's allocated address space.
[EMFILE]	Too many file descriptors are active.
[ENFILE]	No more system file descriptors are available.
[ENFILE]	Insufficient system space to contain i-node.
[ENOSPC]	No more i-nodes are available on the device.

RESTRICTIONS

Should more than 4096 bytes be necessary in any pipe among a loop of processes, deadlock will occur.

SEE ALSO

sh(1), read(2), write(2), fork(2)

PIPE(2)

ASSEMBLER

```
     (pipe = 42.)
     sys pipe
     (read file descriptor in r0)
     (write file descriptor in r1)
```

NAME

profil - execution time profile

SYNOPSIS

```
profil(buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;
```

DESCRIPTION

Buff points to an area of core whose length (in bytes) is given by bufsiz. After this call, the user's program counter (pc) is examined each clock tick (60th second); offset is subtracted from it, and the result multiplied by scale. If the resulting number corresponds to a word inside buff, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777(8) gives a 1-1 mapping of pc's to words in buff; 077777(8) maps each pair of instruction words together. 02(8) 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 exec is executed, but remains on in child and parent both after a fork. Profiling may be turned off if an update in buff would cause a memory fault.

SEE ALSO

monitor(3), prof(1)

ASSEMBLER

```
(profil = 44.)
sys profil; buff; bufsiz; offset; scale
```

PTRACE(2)

NAME

ptrace - process trace

SYNOPSIS

```
#include <signal.h>

ptrace(request, pid, addr, data)
int *addr;
```

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging. There are four arguments whose interpretation depends on a request argument. Generally, pid is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like 'illegal instruction' or externally generated like 'interrupt.' See signal(2) for the list. Then the traced process enters a stopped state and its parent is notified via wait(2). When the child is in the stopped state, its core image can be examined and modified using ptrace. If desired, another ptrace request can then cause the child either to terminate or to continue, possibly ignoring the signal.

The value of the request argument determines the precise action of the call:

0 This request is the only one used by the child 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.

1,2 The word in the child process's address space at addr is returned. If I and D space are separated, request 1 indicates I space, 2 D space. Addr must be even. The child must be stopped. The input data is ignored.

3 The word of the system's per-process data area corresponding to addr is returned. Addr must be even and less than 512. This space contains the registers and other information about the process; its layout corresponds to the user structure in the system.

4,5 The given data is written at the word in the process's address space corresponding to addr, which must be even. No useful value is returned. If I and D space are separated, request 4 indicates I space, 5 D space.

Attempts to write in pure procedure fail if another process is executing the same file.

6 The process's system data is written, as it is read with request 3. 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.

7 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.

8 The traced process terminates.

9 Execution continues as in request 7; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is SIGTRAP. (On the PDP-11 the T-bit is used and just one instruction is executed; on the Interdata the stop does not take place until a store instruction is executed.) This is part of the mechanism for implementing breakpoints.

As indicated, these calls (except for request 0) 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 0177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, ptrace inhibits the set-user-id facility on subsequent exec(2) calls. If a traced process calls exec, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

On the Interdata 8/32, 'word' means a 32-bit word and 'even' means 0 mod 4.

DIAGNOSTICS

[EFAULT] The specified address is out of bounds.

[EPERM] The specified process cannot be traced.

[ESRCH] The specified process does not exist.

RESTRICTIONS

The request 0 call 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 disambiguate.

It should be possible to stop a process on occurrence of a system call; in this way a completely controlled environment could be provided.

SEE ALSO

wait(2), signal(2), adb(1)

ASSEMBLER

```
(ptrace = 26.)
(data in r0)
sys ptrace; pid; addr; request
(value in r0)
```

NAME

read - read from file

SYNOPSIS

```
read(fildes, buffer, nbytes)
char *buffer;
unsigned nbytes;
```

DESCRIPTION

A file descriptor is a word returned from a successful open, creat, dup, or pipe call. Buffer is the location of nbytes contiguous bytes into which the input will be placed. It is not guaranteed that all nbytes bytes will be read; for example if the file refers to a typewriter at most one line will be returned. In any event the number of characters read is returned.

If the returned value is 0, then end-of-file has been reached.

DIAGNOSTICS

Read will fail if one or more of the following is true:

[EBADF]	Fildes is not a valid file descriptor open for reading.
[EFAULT]	Buf points outside the process's allocated address space.
[EFAULT]	The specified address is either odd or out of range when dealing with character special files.
[EINTR]	A read from a slow device was interrupted (before any data arrived) by the delivery of a signal.
[ENXIO]	The /dev/mem address specified is illegal.
[ETPL]	The magtape tape position was lost or the tape unit went offline during a read.
[EWOULDBLOCK]	The process would hang waiting for input (when mode is set to interrupt on each character input).

SEE ALSO

open(2), creat(2), dup(2), pipe(2)

ASSEMBLER

```
(read = 3.)
(file descriptor in r0)
```

READ(2)

```
sys read; buffer; nbytes
(byte count in r0)
```

NAME

renice - set program priority

SYNOPSIS

renice(pid,nice)

DESCRIPTION

Renice sets the scheduling priority of the process with id pid to nice. The superuser may renice any process to any value from -127 to 127. Other users may only increase the nice value of one of their own processes.

DIAGNOSTICS

Renice will fail and the process's scheduling priority will be unaltered if:

[EPERM] The named process's effective user ID is not the same as the calling process and the process's effective user ID is not the superuser.

[ESRCH] The named process does not exist.

SEE ALSO

nice(1), renice(1), nice(2)

ASSEMBLER

(renice = 87.)
sys renice; pid; nice
(old nice value in r0)

NAME

setpgrp, getpgrp - set/get process group

SYNOPSIS

```
int getpgrp(pid)

setpgrp(pid, pgrp)

cc ... -ljobs
```

DESCRIPTION

The process group of the specified process is returned by getpgrp. Setpgrp sets the process group of the specified process pid to the specified pgrp. If pid is zero, then the call applies to the current process.

If the invoker is not the superuser, then the affected process must have the same effective user-id as the invoker or be a descendant of the invoking process.

This call is used by csh(1) to create process groups in implementing job control. The TIOCGPGRP and TIOCSPGRP calls described in tty(4) are used to get/set the process group of the control terminal.

See intro(3J) for a general discussion of job control.

DIAGNOSTICS

Setpgrp will fail and the process group will not be altered if one of the following is true:

[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.

[ESRCH] The requested process does not exist.

RESTRICTIONS

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of these system calls and even the calls themselves are thus subject to change.

A system call setpgrp has been implemented in other versions of UNIX which are not widely used outside of Bell Laboratories; these implementations have, in general, slightly different semantics.

SEE ALSO
csh(1), getuid(2), intro(3J), tty(4)

ASSEMBLER
(setpgrp = 39.)
(process id in r0)
sys setpgrp; newgrp

(getpgrp is implemented as setpgrp(pid,-1))
sys setgrp; -1
(process group in r0)

NAME

setuid, setgid - set user and group ID

SYNOPSIS

```
setuid(uid)

setgid(gid)
```

DESCRIPTION

The user ID (group ID) of the current process is set to the argument. Both the effective and the real ID are set. These calls are only permitted to the superuser or if the argument is the real ID.

DIAGNOSTICS

Setuid will fail and the user ID will be unchanged if:

[EPERM] A non-superuser attempt is made to change the effective user ID to anything other than the real user ID.

Setgid will fail and the group ID will be unchanged if:

[EPERM] A non-superuser attempt is made to change the effective group ID to anything other than the real group ID.

SEE ALSO

getuid(2)

ASSEMBLER

```
(setuid = 23.)
(user ID in r0)
sys setuid

(setgid = 46.)
(group ID in r0)
sys setgid
```

NAME

signal - catch or ignore signals

SYNOPSIS

```
#include <signal.h>

(*signal(sig, func))()
(*func)();
```

DESCRIPTION

A signal is generated by some abnormal event, initiated either by user at a typewriter (quit, interrupt), by a program error (bus error, etc.), or by request of another program (kill). Normally all signals cause termination of the receiving process, but a signal call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals with names as in the include file.

```
SIGHUP  1    hangup
SIGINT  2    interrupt
SIGQUIT 3*   quit
SIGILL  4*   illegal instruction (not reset when caught)
SIGTRAP 5*   trace trap (not reset when caught)
SIGIOT  6*   IOT instruction
SIGEMT  7*   EMT instruction
SIGFPE  8*   floating point exception
SIGKILL 9    kill (cannot be caught or ignored)
SIGBUS  10*  bus error
SIGSEGV 11*  segmentation violation
SIGSYS  12*  bad argument to system call
SIGPIPE 13   write on a pipe or link with no one to read it
SIGALRM 14   alarm clock
SIGTERM 15   software termination signal
        16   unassigned
```

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, sometimes with a core image. If func is SIG_IGN the signal is ignored. Otherwise when the signal occurs func will be called with the signal number as argument. A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal is reset to SIG_DFL after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another signal call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a read or write(2) on a slow device (like a

typewriter; but not a file); and during pause or wait(2). When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call.

The value of signal is the previous (or initial) value of func for the particular signal.

After a fork(2) the child inherits all signals. Exec(2) resets all caught signals to default action.

DIAGNOSTICS

Signal will fail if:

[EINVAL] Sig is an illegal signal number, including SIGKILL.

RESTRICTIONS

If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

The type specification of the routine and its func argument are problematical.

SEE ALSO

kill(1), kill(2), ptrace(2), sigsys(2J), setjmp(3), sigset(3J)

ASSEMBLER

(signal = 48.)
sys signal; sig; label
(old label in r0)

If label is 0, default action is reinstated. If label is odd, the signal is ignored. Any other even label specifies an address in the process where an interrupt is simulated. An RTI or RTT instruction will return from the interrupt.

NAME

sigsys - catch or ignore signals

SYNOPSIS

```
#include <signal.h>

(*sigsys(sig, func))()
void (*func)();
int (*func)(); (if void is not supported)

cc ... -ljobs
```

DESCRIPTION

N.B.: The system currently supports two signal implementations. The one described in signal(2) is standard in version 7 UNIX systems, and retained for backward compatibility as it is different in a number of ways. The one described here (with the interface in sigset(3J)) provides for the needs of the job control mechanisms (see intro(3J)) used by csh(1), and corrects the bugs in the standard implementation of signals, allowing programs which process interrupts to be written reliably.

The routine sigsys is not normally called directly; rather the routines of sigset(3J) should be used. These routines are kept in the 'jobs' library, accessible by giving the loader option -ljobs. The features described here are less portable then those of signal(2) and should not be used in programs which are to be moved to other versions of UNIX.

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 tty(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 which cannot be blocked, the sigsys call allows signals either to be ignored, held until a later time (protecting critical sections in the process), or to cause an interrupt to a specified location. Here is the list of all signals with names as in the include file.

```
SIGHUP  1    hangup
SIGINT  2    interrupt
SIGQUIT 3*   quit
```

SIGILL	4*	illegal instruction (not reset when caught)
SIGTRAP	5*	trace trap (not reset when caught)
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught, held or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
	16	unassigned
SIGSTOP	17†	stop (cannot be caught, held or ignored)
SIGTSTP	18†	stop signal generated from keyboard
SIGCONT	19o	continue after stop
SIGCHLD	20o	child status has changed
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGTINT	23o	input record is available at control terminal
SIGXCPU	24	cpu time limit exceeded (see vlimit(2)) (VAX-11 only)
SIGXFSZ	25	file size limit exceeded (see vlimit(2)) (VAX-11 only)

The starred signals in the list above cause a core image if not caught, held 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 o or †. Signals marked with o are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If func is SIG_HOLD the signal is remembered if it occurs, but not presented to the process; it may be presented later if the process changes the action for the signal. If func is SIG_IGN the signal is subsequently ignored, and pending instances of the signal are discarded (i.e. if the action was previously SIG_HOLD.) Otherwise when the signal occurs func will be called.

A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal, set with sigsys, is reset to SIG_DFL after being caught. However by specifying DEFERSIG(func) as the last argument to sigsys, one causes the action to be set to SIG_HOLD before the interrupt is taken, so that recursive instances of the signal cannot occur during handling of the signal.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a read or write(2) on a slow device (like a terminal;

but not a file) and during a pause or wait(2). When a signal occurs during one of these calls, the saved user status is arranged in such a way that, when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call. Read and write calls which have done no I/O, ioctls blocked with SIGTTOU, and wait2 or wait3 calls are restarted.

The value of sigsys is the previous (or initial) value of func for the particular signal.

The system provides two other functions by oring bits into the signal number: SIGDOPAUSE causes the process to pause after changing the signal action. It can be used to atomically re-enable a held signal which was being processed and wait for another instance of the signal. SIGDORTI causes the system to simulate an rei instruction clearing the mark the system placed on the stack at the point of interrupt before checking for further signals to be presented due to the specified change in signal actions. This allows a signal package such as sigset(3J) to dismiss from interrupts cleanly removing the old state from the stack before another instance of the interrupt is presented.

After a fork(2) or vfork(2) the child inherits all signals. Exec(2) resets all caught signals to default action; held signals remain held and ignored signals remain ignored.

RETURN VALUE

The value BADSIG is returned if the given signal is out of range.

DIAGNOSTICS

Sigsys will fail if:

[EINVAL] Sig is an illegal signal number, including SIGKILL and SIGSTOP.

RESTRICTIONS

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of this facility and the system calls supporting it are thus subject to change.

Since only one signal action can be changed at a time, it is not possible to get the effect of SIGDOPAUSE for more than one signal at a time.

The traps (listed below) should be distinguishable by extra arguments to the signal handler, and all hardware supplied parameters should be made available to the signal routine.

SEE ALSO

kill(1), kill(2), ptrace(2), intro(3J), sigset(3J), setjmp(3), tty(4)

ASSEMBLER (PDP-11)

(signal = 48.)
sys signal; sig; label
(old label in r0)

If label is 0, default action is reinstated. If label is 1, the signal is ignored. If label is 3, the signal is held. Any other even label specifies an address in the process where an interrupt is simulated. If label is otherwise odd, the signal is sent to the function whose address is the label with the low bit cleared with the action set to SIG_HOLD. (Thus DEFERSIG is indicated by the low bit of a signal catch address. An RTI or RTT instruction will return from the interrupt.)

NOTES (VAX-11)

The handler routine can be declared:

```
handler(signo, param, xx, pc, psl)
```

Here signo is the signal name, into which the hardware faults and traps are mapped as defined below. Param is the parameter which is either a constant as given below or, for compatibility mode faults, the code provided by the hardware. Compatibility mode faults are distinguished from the other SIGILL traps by having PSL_CM set in the psl.

The routine is actually called with only 3 parameters specified in the calls or callg instruction. After return from the signal handler the pc and psl are popped off of the stack with an rei, so they act as 'value-result' parameters unlike normal C value parameters.

The following defines the mapping of hardware traps to signals and codes. All of these symbols are defined in <signal.h>:

Hardware condition	Signal	Code
Arithmetic traps:		
Integer overflow	SIGFPE	FPE_INTOVF_TRAP
Integer division by zero	SIGFPE	FPE_INTDIV_TRAP
Floating overflow trap	SIGFPE	FPE_FLTOVF_TRAP
Floating/decimal division	SIGFPE	FPE_FLTDIV_TRAP

by zero		
Floating underflow trap	SIGFPE	FPE_FLTUND_TRAP
Decimal overflow trap	SIGFPE	FPE_DECOVF_TRAP
Subscript-range	SIGFPE	FPE_SUBRNG_TRAP
Floating overflow fault	SIGFPE	FPE_FLTOVF_FAULT
Floating divide by zero fault	SIGFPE	FPE_FLTDIV_FAULT
Floating underflow fault	SIGFPE	FPE_FLTUND_FAULT
Length access control	SIGSEGV	
Protection violation	SIGBUS	
Reserved instruction	SIGILL	ILL_RESAD_FAULT
Customer-reserved instr.	SIGEMT	
Reserved operand	SIGILL	ILL_PRIVIN_FAULT
Reserved addressing	SIGILL	ILL_RESOP_FAULT
Trace pending	SIGTRAP	
Bpt instruction	SIGTRAP	
Compatibility-mode	SIGILL	hardware supplied code
Chme	SIGSEGV	
Chms	SIGSEGV	
Chmu	SIGSEGV	

NAME

stat, fstat - get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

stat(name, buf)
char *name;
struct stat *buf;

fstat(fildes, buf)
struct stat *buf;
```

DESCRIPTION

Stat obtains detailed information about a named file. Fstat obtains the same information about an open file known by the file descriptor from a successful open, creat, dup or pipe(2) call.

Name points to a null-terminated string naming a file; buf is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be searchable. The layout of the structure pointed to by buf as defined in <stat.h> is given below. St mode is encoded according to the '#define' statements.

```
/*
 * SCCSID: @(#)stat.h    1.0  11/12/83
 */
struct    stat
{
     dev_t     st_dev;
     ino_t     st_ino;
     unsigned  short st_mode;
     short     st_nlink;
     short     st_uid;
     short     st_gid;
     dev_t     st_rdev;
     off_t     st_size;
     time_t    st_atime;
     time_t    st_mtime;
     time_t    st_ctime;
};

#define   S_IFMT     0170000  /* type of file */
#define   S_IFDIR    0040000  /* directory */
#define   S_IFCHR    0020000  /* character special */
#define   S_IFBLK    0060000  /* block special */
#define   S_IFREG    0100000  /* regular */
```

```
#define   S_IFMPC    0030000  /* multiplexed char special */
#define   S_IFMPB    0070000  /* multiplexed block special */
#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)). The defined types, ino t, off t, time t, name various width integer values; dev t encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h> (see types(5).

When fildes is associated with a pipe, fstat reports an ordinary file with restricted permissions. The size is the number of bytes queued in the pipe.

st atime is the file was last read. For reasons of efficiency, it is not set when a directory is searched, although this would be more logical. st mtime is the time the file was last written or created. It is not set by changes of owner, group, link count, or mode. st ctime is set both by writing and changing the i-node.

DIAGNOSTICS

Stat will fail if one or more of the following is true:

[EACCES] Search permission is denied for a component of the path prefix.

[EFAULT] Name or buf points to an invalid address.

[ENFILE] Insufficient system space to contain i-node.

[ENOENT] The named file, or an element within the named file, does not exist.

[ENOTDIR] A component of the path prefix is not a directory.

Fstat will fail if one or both of the following is true:

[EBADF] Fildes is not a valid open file descriptor.

[EFAULT] Buf points to an address outside of the process's allocated address space.

SEE ALSO
ls(1), filsys(5)

ASSEMBLER
(stat = 18.)
sys stat; name; buf

(fstat = 28.)
(file descriptor in r0)
sys fstat; buf

NAME

stime - set time

SYNOPSIS

```
stime(tp)
long *tp;
```

DESCRIPTION

Stime sets the system's idea of the time and date. Time, pointed to by tp, is measured in seconds from 0000 GMT Jan 1, 1970. Only the superuser may use this call.

DIAGNOSTICS

Stime will fail and the system's current time will be unchanged if:

[EPERM] The process's effective user ID is not the superuser.

SEE ALSO

date(1), time(2), ctime(3)

ASSEMBLER

```
(stime = 25.)
(time in r0-r1)
sys stime
```

SYNC(2)

NAME

sync - update super-block

SYNOPSIS

sync( )

DESCRIPTION

Sync causes all information in core memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

It should be used by programs which examine a file system, for example icheck, df, etc. It is mandatory before a boot.

SEE ALSO

sync(1), update(8)

RESTRICTIONS

The writing, although scheduled, is not necessarily complete upon return from sync.

ASSEMBLER

(sync = 36.)
sys sync

NAME

time, ftime - get date and time

SYNOPSIS

```
long time(0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;
```

DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If tloc is nonnull, 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>:

```
/*
 * SCCSID: @(#)timeb.h1.011/12/83
 */
/*
 * Structure returned by ftime system call
 */
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 timezone (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.

DIAGNOSTICS

Ftime will fail if:

[EFAULT] Tp points to an address outside the process's allocated address space.

SEE ALSO

date(1), stime(2), ctime(3)

```
ASSEMBLER
     (ftime = 35.)
     sys ftime; bufptr

     (time = 13.; obsolete call)
     sys time
     (time since 1970 in r0-r1)
```

NAME

times - get process times

SYNOPSIS

```
times(buffer)
struct tbuffer *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=60 in North America.

After the call, the buffer will appear as follows:

```
struct tbuffer {
     long proc_user_time;
     long proc_system_time;
     long child_user_time;
     long child_system_time;
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO

time(1), time(2)

ASSEMBLER

```
(times = 43.)
sys times; buffer
```

NAME

umask - set file creation mode mask

SYNOPSIS

umask(complmode)

DESCRIPTION

Umask sets a mask used whenever a file is created by creat(2) or mknod(2): the actual mode (see chmod(2)) of the newly-created file is the logical and of the given mode and the complement of the argument. Only the low-order 9 bits of the mask (the protection bits) participate. In other words, the mask shows the bits to be turned off when files are created.

The previous value of the mask is returned by the call. The value is initially 0 (no restrictions). The mask is inherited by child processes.

SEE ALSO

creat(2), mknod(2), chmod(2)

ASSEMBLER

```
(umask = 60.)
sys umask; complmode
```

NAME

unlink - remove directory entry

SYNOPSIS

```
unlink(name)
char *name;
```

DESCRIPTION

Name points to a null-terminated string. Unlink removes the entry for the file pointed to by name from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

DIAGNOSTICS

The unlink will succeed unless:

[EACCES]	Search permission is denied for a component of the path prefix.
[EACCES]	Write permission is denied on the directory containing the link to be removed.
[EBUSY]	The entry to be unlinked is the mount point for a mounted file system.
[EFAULT]	Name points to an address outside the process's allocated address space.
[ENFILE]	Insufficient system space to contain i-node.
[ENOENT]	The named file, or an element within the named file, does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.
[EPERM]	The named file is a directory and the process's effective user ID is not the superuser.
[EROFS]	The named file resides on a read-only file system.
[ETXTBSY]	The file is a pure procedure (shared text) file that is currently open for reading or writing by some process.

SEE ALSO

rm(1), link(2)

ASSEMBLER

```
     (unlink = 10.)
     sys unlink; name
```

NAME

utime - set file times

SYNOPSIS

```
#include <sys/types.h>
utime(file, timep)
char *file;
time_t timep[2];
```

DESCRIPTION

The utime call uses the 'accessed' and 'updated' times in that order from the timep vector to set the corresponding recorded times for file.

The caller must be the owner of the file or the superuser. The 'inode-changed' time of the file is set to the current time.

DIAGNOSTICS

Utime will fail if one or more of the following is true:

[EACCES] A component of the path prefix denies search permission.

[EFAULT] Timep points to an address outside the process's allocated address space.

[ENFILE] Insufficient system space to contain i-node.

[ENOENT] The named file, or an element within the named file, does not exist.

[ENOTDIR] A component of the path prefix is not a directory.

[EPERM] The process's effective user ID is not the superuser and not the same as the owner of the file.

[EROFS] The named file is located on a read-only file system.

SEE ALSO

stat (2)

ASSEMBLER

```
(utime = 30.)
sys utime; file; timep
```

NAME

wait - wait for process to terminate

SYNOPSIS

```
wait(status)
int *status;

wait(0)
```

DESCRIPTION

Wait causes its caller to delay until a signal is received or one of its child processes terminates. If any child has died since the last wait, return is immediate; if there are no children, return is immediate with the error bit set (resp. with a value of -1 returned). The normal return yields the process ID of the terminated child. In the case of several children several wait calls are needed to learn of all the deaths.

If (int)status is nonzero, the high byte of the word pointed to receives the low byte of the argument of exit when the child terminated. The low byte receives the termination status of the process. See signal(2) for a list of termination statuses (signals); 0 status indicates normal termination. A special status (0177) is returned for a stopped process which has not terminated and can be restarted. See ptrace(2). If the 0200 bit of the termination status is set, a core image of the process was produced by the system.

If the parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

RETURN VALUE

If wait returns due to the receipt of a signal, a value of -1 is returned to the calling process and the errno is set to EINTR. 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.

DIAGNOSTICS

Wait will fail and return immediately if the following is true:

[ECHILD] The calling process has no unwaited-for child processes.

[EFAULT] The status argument points to an address outside of the process's allocated address space.

SEE ALSO
exit(2), fork(2), signal(2)

ASSEMBLER
(wait = 7.)
sys wait
(process ID in r0)
(status in r1)

The high byte of the status is the low byte of r0 in the child at termination.

NAME

wait2 - wait for process to terminate

SYNOPSIS

```
#include <wait.h>

wait2(&w.w_status, options)
union wait w;
int options;

cc ... -ljobs
```

DESCRIPTION

Wait2 is similar to the standard wait(2) system call, but allow additional options useful with job control. They return the process ID of a terminated or stopped child process. The w.w_status and option words are described by definitions and macros in the file <wait.h>; the union and its bitfield definitions and associated macros given there provide convenient and mnemonic access to the word of status returned by a wait2 call. If the call returns a process ID, several macros are available to interpret the status word returned. If the process is stopped, WIFSTOPPED(w) is true, and the signal that caused it to stop is w.w_stopsig. If the process is not stopped (has terminated), WIFEXITED(w) determines whether it terminated by calling exit(2); if so, the exit code is w.w_retcode. WIFSIGNALED(w) is true if the process was terminated by a signal (see signal(2)); the signal causing termination was w.w_termsig, and w.w_coredump indicates whether a core dump was produced.

There are two options, which may be combined by oring them together. The first is WNOHANG which causes the wait2 to not hang if there are no processes which wish to report status, rather returning a pid of 0 in this case as the result of the wait2. The second option is WUNTRACED which causes wait2 to return information when children of the current process which are stopped but not traced (with ptrace(2)) because they received a SIGTTIN, SIGTTOU, SIGTSTP or SIGSTOP signal. See sigsys(2J) for a description of these signals.

RETURN VALUE

Returns -1 if there are no children not previously waited for, or 0 if the WNOHANG option is given and there are no stopped or exited children.

DIAGNOSTICS

Wait2 will fail and return immediately if one or more of the following is true:

[ECHILD] The calling process has no unwaited-for child

processes.

[EFAULT] The w.w_status argument points to an address outside the process's allocated address space.

RESTRICTIONS

This call is peculiar to this version of UNIX. The options and specifications of this system call and even the call itself are subject to change. It may be replaced by other facilities in future versions of the system.

SEE ALSO

exit(2), fork(2), sigsys(2J), wait(2)

NAME

write - write on a file

SYNOPSIS

```
write(fildes, buffer, nbytes)
char *buffer;
```

DESCRIPTION

A file descriptor is a word returned from a successful open, creat, dup, or pipe(2) call.

Buffer is the address of nbytes contiguous bytes which are written on the output file. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Writes which are multiples of 512 characters long and begin on a 512-byte boundary in the file are more efficient than any others.

DIAGNOSTICS

Write will fail if one or more of the following is true:

[EBADF]	Fildes is not a valid descriptor open for writing.
[EFAULT]	The specified address is odd or the count is odd when dealing with disk character special files.
[EFAULT]	The buffer points to an address outside the process's allocated address space.
[EFBIG]	An attempt was made to write a file that exceeds the maximum file size.
[ENXIO]	The specified /dev/mem address is illegal.
[EPIPE]	An attempt is made to write to a pipe that is not open for reading by any process.
[EQUOT]	An attempt was made to write a file that exceeds a quota governing the file.
[ETPL]	The magtape tape position was lost or the tape went offline during a write.

SEE ALSO

creat(2), open(2), pipe(2)

ASSEMBLER

(write = 4.)

WRITE(2)

```
(file descriptor in r0)
sys write; buffer; nbytes
(byte count in r0)
```

NAME

zaptty - zap the controlling tty

SYNOPSIS

zaptty( )

DESCRIPTION

Zaptty will disassociates a process from it's controlling tty. The next terminal type device that the process opens will become the new controlling tty. Zaptty is useful for daemons, since they usually do not want to be associated with any terminal.

For obvious reasons, this call is restricted to the superuser.

ASSEMBLER

```
(zaptty = 69.)
sys zaptty
```

NAME

intro - introduction to library functions

SYNOPSIS

```
#include <stdio.h>

#include <math.h>
```

DESCRIPTION

This section describes functions that may be found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in section 2. Functions are divided into various libraries distinguished by the section number at the top of the page:

(3) These functions, together with those of section 2 and those marked (3S), constitute librarys libc and libcov, which is automatically loaded by the C compiler cc(1) and the Fortran compiler f77(1). The link editor ld(1) searches this library under the '-lc' or '-lcov' option. Declarations for some of these functions may be obtained from include files indicated on the appropriate pages.

(3M) These functions constitute the math library, libm. They are automatically loaded as needed by the Fortran compiler f77(1). The link editor searches this library under the '-lm' option. Declarations for these functions may be obtained from the include file <math.h>.

(3S) These functions constitute the 'standard I/O package', see stdio(3). These functions are in the librarys libc and libcov already mentioned. Declarations for these functions may be obtained from the include file <stdio.h>.

(3X) Various specialized libraries have not been given distinctive captions. The files in which these libraries are found are named on the appropriate pages.

FILES

/lib/libc.a
/lib/libcov.a
/lib/libm.a, /usr/lib/libm.a (one or the other)

SEE ALSO

stdio(3), nm(1), ld(1), cc(1), f77(1), intro(2)

DIAGNOSTICS

Functions in the math library (3M) may return conventional values when the function is undefined for the given

INTRO(3)

arguments or when the value is not representable. In these cases the external variable errno (see intro(2)) is set to the value EDOM or ERANGE. The values of EDOM and ERANGE are defined in the include file <math.h>.

ASSEMBLER

In assembly language these functions may be accessed by simulating the C calling sequence. For example, ecvt(3) might be called this way:

```
setd
mov  $sign,-(sp)
mov  $decpt,-(sp)
mov  ndigit,-(sp)
movf value,-(sp)
jsr  pc,_ecvt
add  $14.,sp
```

NAME

intro - summary of job control facilities

SYNOPSIS

```
#include <sys/ioctl.h>
#include <signal.h>
#include <wait.h>

int fildes, signo;
short pid, pgrp;
union wait status;

ioctl(fildes, TIOCSPGRP, &pgrp)
ioctl(fildes, TIOCGPGRP, &pgrp)

sigset(signo, action)
sighold(signo)
sigrelse(signo)
sigpause(signo)
sigsys(signo, action)

wait2(&status, options)

cc ... -ljobs
```

DESCRIPTION

The facilities described here are used to support the job control implemented in csh(1), and may be used in other programs to provide similar facilities. Because these facilities are not standard in UNIX and because the signal mechanisms are also slightly different, the associated routines are not in the standard C library, but rather in the -ljobs library.

For descriptions of the individual routines see the various sections listed in SEE ALSO below. This section attempt only to place these facilities in context, not to explain the semantics of the individual calls.

Terminal arbitration mechanisms.

The job control mechanism works by associating with each process a number called a process group ; related processes (e.g. in a pipeline) are given the same process group. The system assigns a single process group number to each terminal. Processes running on a terminal are given read access to that terminal only if they are in the same process group as that terminal.

Thus a command interpreter may start several jobs running in different process groups and arbitrate access to the terminal by controlling which, if any, of these processes is in

INTRO(3J)

the same process group as the terminal. When a process which is not in the process group of the terminal tries to read from the terminal, all members of the process group of the process receive a SIGTTIN signal, which normally then causes them to stop until they are continued with a SIGCONT signal. (See sigsys(2J) for a description of these signals; tty(4) for a description of process groups.)

If a process which is not in the process group of the terminal attempts to change the terminals mode, the process group of that process is sent a SIGTTOU signal, causing the process group to stop. A similar mechanism is (optionally) available for output, causing processes to block with SIGTTOU when they attempt to write to the terminal while not in its process group; this is controlled by the LTOSTOP bit in the tty mode word, enabled by "stty tostop" and disabled (the default) by "stty -tostop." (The LTOSTOP bit is described in tty(4)).

How the shell manipulates process groups.

A shell which is interactive first establishes its own process group and a process group for the terminal; this prevents other processes from being inadvertently stopped while the terminal is under its control. The shell then assigns each job it creates a distinct process group. When a job is to be run in the foreground, the shell gives the terminal to the process group of the job using the TIOCSPGRP ioctl (See ioctl(2) and tty(4)). When a job stops or completes, the shell reclaims the terminal by resetting the terminals process group to that of the shell using TIOCSPGRP again.

Shells which are running shell scripts or running non-interactively do not manipulate process groups of jobs they create. Instead, they leave the process group of sub-processes and the terminal unchanged. This assures that if any sub-process they create blocks for terminal i/o, the shell and all its sub-processes will be blocked (since they are a single process group). The first interactive parent of the non-interactive shell can then be used to deal with the stoppage.

Processes which are orphans (whose parents have exited), and descendants of these processes are protected by the system from stopping, since there can be no interactive parent. Rather than blocking, reads from the control terminal return end-of-file and writes to the control terminal are permitted (i.e. LTOSTOP has no effect for these processes.) Similarly processes which ignore or hold the SIGTTIN or SIGTTOU signal are not sent these signals when accessing their control terminal; if they are not in the process group of the control

terminal reads simply return end-of-file. Output and mode setting are also allowed.

Before a shell suspends itself, it places itself back in the process group in which it was created, and then sends this original group a stopping signal, stopping the shell and any other intermediate processes back to an interactive parent. The shell also restores the process group of the terminal when it finishes, as the process which then resumes would not necessarily be in control of the terminal otherwise.

Naive processes.

A process which does not alter the state of the terminal, and which does no job control can invoke subprocesses normally without worry. If such a process issues a system(3S) call and this command is then stopped, both of the processes will stop together. Thus simple processes need not worry about job control, even if they have "shell escapes" or invoke other processes.

Processes which modify the terminal state.

When first setting the terminal into an unusual mode, the process should check, with the stopping signals held, that it is in the foreground. It should then change the state of the terminal, and set the catches for SIGTTIN, SIGTTOU and SIGTSTP. The following is a sample of the code that will be needed, assuming that unit 2 is known to be a terminal.

```
short     tpgrp;
...

retry:
     sigset(SIGTSTP, SIG_HOLD);
     sigset(SIGTTIN, SIG_HOLD);
     sigset(SIGTTOU, SIG_HOLD);
     if (ioctl(2, TIOCGPGRP, &tpgrp) != 0)
          goto nottty;
     if (tpgrp != getpgrp(0)) { /* not in foreground */
          sigset(SIGTTOU, SIG_DFL);
          kill(0, SIGTTOU);
          /* job stops here waiting for SIGCONT */
          goto retry;
     }
     ...save old terminal modes and set new modes...
     sigset(SIGTTIN, onstop);
     sigset(SIGTTOU, onstop);
     sigset(SIGTSTP, onstop);
```

It is necessary to ignore SIGTSTP in this code because otherwise our process could be moved from the foreground to the

INTRO(3J)

background in the middle of checking if it is in the foreground. The process holds all the stopping signals in this critical section so no other process in our process group can mess us up by blocking us on one of these signals in the middle of our check. (This code assumes that the command interpreter will not move a process from foreground to background without stopping it; if it did we would have no way of making the check correctly.)

The routine which handles the signal should clear the catch for the stop signal and kill(2) the processes in its process group with the same signal. The statement after this kill will be executed when the process is later continued with SIGCONT.

Thus the code for the catch routine might look like:

```
        ...
        sigset(SIGTSTP, onstop);
        sigset(SIGTTIN, onstop);
        sigset(SIGTTOU, onstop);
        ...

onstop(signo)
        int signo;
{
        ... restore old terminal state ...
        sigset(signo, SIG_DFL);
        kill(0, signo);
        /* stop here until continued */
        sigset(signo, onstop);
        ... restore our special terminal state ...
}
```

This routine can also be used to simulate a stop signal.

If a process does not need to save and restore state when it is stopped, but wishes to be notified when it is continued after a stop it can catch the SIGCONT signal; the SIGCONT handler will be run when the process is continued.

Processes which lock data bases such as the password file should ignore SIGTTIN, SIGTTOU, and SIGTSTP signals while the data bases are being manipulated. While a process is ignoring SIGTTIN signals, reads which would normally have hung will return end-of-file; writes which would normally have caused SIGTTOU signals are instead permitted while SIGTTOU is ignored.

Interrupt-level process handling.

Using the mechanisms of sigset(3J) it is possible to handle process state changes as they occur by providing an interrupt-handling routine for the SIGCHLD signal which occurs whenever the status of a child process changes. A signal handler for this signal is established by:

```
sigset(SIGCHLD, onchild);
```

The shell or other process would then await a change in child status with code of the form:

```
recheck:
     sighold(SIGCHLD);          /* start critical section */
     if (no children to process) {
          sigpause(SIGCHLD);    /* release SIGCHLD and pause */
          goto recheck;
     }
     sigrelse(SIGCHLD);         /* end critical region */
     /* now have a child to process */
```

Here we are using sighold to temporarily block the SIGCHLD signal during the checking of the data structures telling us whether we have a child to process. If we didn't block the signal we would have a race condition since the signal might corrupt our decision by arriving shortly after we had finished checking the condition but before we paused.

If we need to wait for something to happen, we call sigpause which automatically releases the hold on the SIGCHLD signal and waits for a signal to occur by starting a pause(2). Otherwise we simply release the SIGCHLD signal and process the child. Sigpause is similar to the PDP-11 wait instruction, which returns the priority of the processor to the base level and idles waiting for an interrupt.

It is important to note that the long-standing bug in the signal mechanism which would have lost a SIGCHLD signal which occurred while the signal was blocked has been fixed. This is because sighold uses the SIG_HOLD signal set of sigsys(2J) to prevent the signal action from being taken without losing the signal if it occurs. Similarly, a signal action set with sigset has the signal held while the action routine is running, much as a the interrupt priority of the processor is raised when a device interrupt is taken.

In this interrupt driven style of termination processing it is necessary that the wait calls used to retrieve status in the SIGCHLD signal handler not block. This is because a single invocation of the SIGCHLD handler may indicate an arbitrary number of process status changes: signals are not queued. This is similar to the case in a disk driver where several drives on a single controller may report status at

INTRO(3J)

once, while there is only one interrupt taken. It is even possible for no children to be ready to report status when the SIGCHLD handler is invoked, if the signal was posted while the SIGCHLD handler was active, and the child was noticed due to a SIGCHLD initially sent for another process. This causes no problem, since the handler will be called whenever there is work to do; the handler just has to collect all information by calling wait3 until it says no more information is available. Further status changes are guaranteed to be reflected in another SIGCHLD handler call.

Restarting system calls.

In older versions of UNIX "slow" system calls were interrupted when signals occurred, returning EINTR. The new signal mechanism sigset(3J) normally restarts such calls rather than interrupting them. To summarize: pause and wait return error EINTR (as before), ioctl and wait3 restart, and read and write restart unless some data was read or written in which case they return indicating how much data was read or written. In programs which use the older signal(2) mechanisms, all of these calls return EINTR if a signal occurs during the call.

SEE ALSO

csh(1), ioctl(2), killpg(2), setpgrp(2), sigsys(2J), wait2(2J), signal(2), tty(4)

RESTRICTIONS

The job control facilities are not available in standard version 7 UNIX. These facilities are still under development and may change in future releases of the system as better inter-process communication facilities and support for virtual terminals become available. The options and specifications of these system calls and even the calls themselves are thus subject to change.

ABORT(3)

NAME

abort - generate IOT fault

SYNOPSIS

abort()

DESCRIPTION

Abort executes the PDP11 IOT instruction. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

SEE ALSO

adb(1), signal(2), exit(2)

DIAGNOSTICS

Usually 'IOT trap - core dumped' from the shell.

ABS(3)

NAME

abs - integer absolute value

SYNOPSIS

abs(i)

DESCRIPTION

Abs returns the absolute value of its integer operand.

SEE ALSO

floor(3) for fabs

RESTRICTIONS

You get what the hardware gives on the largest negative integer.

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 causes an exit(2) with a diagnostic comment on the standard output when expression is false (0). Compiling with the cc(1) option -DNDEBUG effectively deletes assert from the program.

DIAGNOSTICS

'Assertion failed: file f line n.' F is the source file and n the source line number of the assert statement.

ATOF(3)

NAME

atof, atoi, atol - convert ASCII to numbers

SYNOPSIS

```
double atof(nptr)
char *nptr;

atoi(nptr)
char *nptr;

long atol(nptr)
char *nptr;
```

DESCRIPTION

These functions convert a string pointed to by nptr to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional 'e' or 'E' followed by an optionally signed integer.

Atoi and atol recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3)

RESTRICTIONS

There are no provisions for overflow.

NAME

crypt, encrypt - a one way hashing encryption algorithm

SYNOPSIS

```
char *crypt(key, salt)
char *key, *salt;

encrypt(block)
char *block;
```

DESCRIPTION

Crypt is the password encryption routine. It is based on a one way hashing encryption algorithm with variations intended (among other things) to frustrate use of hardware implementations of a key search

Key is a user's typed password. Salt is a two-character string chosen from the set [a-zA-Z0-9./]; this string is used to perturb the hashing 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. The first two characters are the salt itself.

There is a character array of length 64 containing only the characters with numerical value 0 and 1. When this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine by crypt.

The encrypt entry provides (rather primitive) access to the actual hashing algorithm. The argument to the encrypt entry is a character array of length 64 containing only the characters with numerical value of 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 hashing algorithm using the key set by crypt.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

RESTRICTIONS

The return value points to static data whose content is overwritten by each call.

CTIME(3)

NAME

ctime, localtime, gmtime, asctime, timezone - convert date and time to ASCII

SYNOPSIS

```
char *ctime(clock)
long *clock;

#include <time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

DESCRIPTION

Ctime converts a time pointed to by clock such as returned by time(2) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

Localtime and gmtime return pointers to structures containing the broken-down time. Localtime corrects for the time zone and possible daylight savings time; gmtime converts directly to GMT, which is the time UNIX uses. Asctime converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
/*
 * SCCSID: @(#)time.h    1.0  11/12/83
 */
struct tm { /* see ctime(3) */
        int  tm_sec;
        int  tm_min;
        int  tm_hour;
        int  tm_mday;
        int  tm_mon;
        int  tm_year;
        int  tm_wday;
        int  tm_yday;
        int  tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday = 0), year - 1900, day of year (0-365), and a flag that is nonzero if daylight saving time is in effect.

When local time is called for, the program consults the system to determine the time zone and whether the standard U.S.A. daylight saving time adjustment is appropriate. The program knows about the peculiarities of this conversion in 1974 and 1975; if necessary, a table for these years can be extended.

Timezone returns 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 Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; e.g. 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.

SEE ALSO
time(2)

RESTRICTIONS
The return values point to static data whose content is overwritten by each call.

CTYPE(3)

NAME

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii - character classification

SYNOPSIS

```
#include <ctype.h>

isalpha(c)

. . .
```

DESCRIPTION

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 is true and on the single non-ASCII value EOF (see stdio(3)).

isalpha	c is a letter
isupper	c is an upper case letter
islower	c is a lower case letter
isdigit	c is a digit
isalnum	c is an alphanumeric character
isspace	c is a space, tab, carriage return, newline, or formfeed
ispunct	c is a punctuation character (neither control nor alphanumeric)
isprint	c is a printing character, code 040(8) (space) through 0176 (tilde)
iscntrl	c is a delete character (0177) or ordinary control character (less than 040).
isascii	c is an ASCII character, code less than 0200

SEE ALSO

ascii(7)

NAME

curses - screen functions with "optimal" cursor motion

SYNOPSIS

cc [flags] files -lcurses -ltermlib [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

Screen Updating and Cursor Movement Optimization: A Library Package, Ken Arnold,
stty(2), setenv(3), termcap(5)

AUTHOR

Ken Arnold

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
crmode()	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
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

CURSES(3)

```
newwin(lines,cols,begin_y,begin_x) create a new window
nl()                               set newline mapping
nocrmode()                         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
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 char 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

dbminit, fetch, store, delete, firstkey, nextkey - data base subroutines

SYNOPSIS

```
typedef struct { char *dptr; int dsize; } datum;

dbminit(file)
char *file;

datum fetch(key)
datum key;

store(key, content)
datum key, content;

delete(key)
datum key;

datum firstkey();

datum nextkey(key);
datum key;
```

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 filesystem accesses. The functions are obtained with the loader option -ldbm.

Keys and contents 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.)

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:

DBM(3X)

```
for(key=firstkey(); key.dptr!=NULL; key=nextkey(key))
```

DIAGNOSTICS

All functions that return an int indicate errors with negative values. A zero return indicates ok. Routines that return a datum indicate errors with a null (0) dptr.

RESTRICTIONS

The '.pag' file will contain holes so that its apparent size is about four times its actual content. Older UNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) 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 512 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.

NAME

ecvt, fcvt, gcvt - output conversion

SYNOPSIS

```
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;
char *buf;
```

DESCRIPTION

Ecvt converts the value to a null-terminated string of ndigit ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through decpt (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by sign is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to ecvt, except that the correct di has been rounded for Fortran F-format output of the number of digits specified by ndigits.

Gcvt converts the value to a null-terminated ASCII string in buf and returns a pointer to buf. It attempts to produce ndigit significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3)

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

END(3)

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 coincides with end, but many functions reset the program break, among them the routines of brk(2), malloc(3), standard input/output (stdio(3)), the profile (-p) option of cc(1), etc. The current value of the program break is reliably returned by 'sbrk(0)', see brk(2).

SEE ALSO

brk(2), malloc(3)

NAME

exp, log, log10, pow, sqrt - exponential, logarithm, power, square root

SYNOPSIS

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

DESCRIPTION

Exp returns the exponential function of x.

Log returns the natural logarithm of x; log10 returns the base 10 logarithm.

Pow returns x .

Sqrt returns the square root of x.

SEE ALSO

hypot(3), sinh(3), intro(2)

DIAGNOSTICS

Exp and pow return a huge value when the correct value would overflow; errno is set to ERANGE. Pow returns 0 and sets errno to EDOM when the second argument is negative and non-integral and when both arguments are 0.

Log returns 0 when x is zero or negative; errno is set to EDOM.

Sqrt returns 0 when x is negative; errno is set to EDOM.

FCLOSE(3S)

NAME

fclose, fflush - close or flush a stream

SYNOPSIS

```
#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;
```

DESCRIPTION

Fclose causes any buffers for the named stream to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling exit(2).

Fflush causes any buffered data for the named output stream to be written to that file. The stream remains open.

SEE ALSO

close(2), fopen(3), setbuf(3)

DIAGNOSTICS

These routines return EOF if stream is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

feof, ferror, clearerr, fileno - stream status inquiries

SYNOPSIS

```
#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream;
```

DESCRIPTION

Feof returns non-zero when end of file is read on the named input stream, otherwise zero.

Ferror returns non-zero when an error has occurred reading or writing the named stream, otherwise zero. Unless cleared by clearerr, the error indication lasts until the stream is closed.

Clrerr resets the error indication on the named stream.

Fileno returns the integer file descriptor associated with the stream, see open(2).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3), open(2)

FLOOR(3M)

NAME

fabs, floor, ceil - absolute value, floor, ceiling functions

SYNOPSIS

```
#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double(x);
```

DESCRIPTION

Fabs returns the absolute value |x|.

Floor returns the largest integer not greater than x.

Ceil returns the smallest integer not less than x.

SEE ALSO

abs(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. Fopen returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r" open for reading

"w" create for writing

"a" append: open for writing at end of file, or create for writing

Freopen substitutes the named file in place of the open stream. It returns the original value of stream. The original stream is closed.

Freopen is typically used to attach the preopened constant names, stdin, stdout, stderr, to specified files.

Fdopen associates a stream with a file descriptor obtained from open, dup, creat, or pipe(2). The type of the stream must agree with the mode of the open file.

SEE ALSO

open(2), fclose(3)

DIAGNOSTICS

Fopen and freopen return the pointer NULL if filename cannot be accessed.

RESTRICTIONS

Fdopen is not portable to systems other than UNIX.

FREAD(3S)

NAME

fread, fwrite - buffered binary input/output

SYNOPSIS

```
#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

DESCRIPTION

Fread reads, into a block beginning at ptr, nitems of data of the type of *ptr from the named input stream. It returns the number of items actually read.

Fwrite appends at most nitems of data of the type of *ptr beginning at ptr to the named output stream. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3), getc(3), putc(3), gets(3), puts(3), printf(3), scanf(3)

DIAGNOSTICS

Fread and fwrite return 0 upon end of file or error.

NAME

frexp, ldexp, modf - split into mantissa and exponent

SYNOPSIS

```
double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;

double modf(value, iptr)
double value, *iptr;
```

DESCRIPTION

Frexp returns the mantissa of a double value as a double quantity, x, of magnitude less than 1 and stores an integer n such that value = x*2**n indirectly through eptr.

Ldexp returns the quantity value*2**exp.

Modf returns the positive fractional part of value and stores the integer part indirectly through iptr.

FSEEK(3S)

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)
```

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.

Fseek undoes any effects of ungetc(3).

Ftell returns the current value of the offset relative to the beginning of the file associated with the named stream. It is measured in bytes on UNIX; on some other systems it is a magic cookie, and the only foolproof way to obtain an offset for fseek.

Rewind(stream) is equivalent to fseek(stream, 0L, 0).

SEE ALSO

lseek(2), fopen(3)

DIAGNOSTICS

Fseek returns -1 for improper seeks.

NAME

getc, getchar, fgetc, getw - get character or word 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 from the named input stream.

Getchar() is identical to getc(stdin).

Fgetc behaves like getc, but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next word from the named input stream. It returns the constant EOF upon end of file or error, but since that is a good integer value, feof and ferror(3) should be used to check the success of getw. Getw assumes no special alignment in the file.

SEE ALSO

fopen(3), putc(3), gets(3), scanf(3), fread(3), ungetc(3)

DIAGNOSTICS

These functions return the integer constant EOF at end of file or upon read error.

A stop with message, 'Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by fopen.

RESTRICTIONS

The end-of-file return from getchar is incompatible with that in UNIX editions 1-6.

Because it is implemented as a macro, getc treats a stream argument with side effects incorrectly. In particular, 'getc(*f++);' doesn't work sensibly.

GETENV(3)

NAME

getenv - value for environment name

SYNOPSIS

```
char *getenv(name)
char *name;
```

DESCRIPTION

Getenv searches the environment list (see environ(5)) for a string of the form name=value and returns value if such a string is present, otherwise 0 (NULL).

SEE ALSO

environ(5), exec(2)

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent - get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent();

struct group *getgrgid(gid) int gid;

struct group *getgrnam(name) char *name;

int setgrent();

int endgrent();
```

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.

```
/*
 * SCCSID: @(#)grp.h     1.0  11/12/83
 */
struct    group { /* see getgrent(3) */
     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
: Null-terminated vector of pointers to the individual member names.

Getgrent simply reads the next line while getgrgid and getgrnam search until a matching gid or name is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to setgrent 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.

FILES

/etc/group

SEE ALSO

getlogin(3), getpwent(3), group(5)

DIAGNOSTICS

A null pointer (0) is returned on EOF or error.

RESTRICTIONS

All information is contained in a static area so it must be copied if it is to be saved.

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 userid is shared by several login names.

If getlogin is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to first call getlogin and if it fails, to call getpwuid.

FILES

/etc/utmp

SEE ALSO

getpwent(3), getgrent(3), utmp(5)

DIAGNOSTICS

Returns NULL (0) if name not found.

RESTRICTIONS

The return values point to static data whose content is overwritten by each call.

NAME

getpass - read a password

SYNOPSIS

```
char *getpass(prompt)
char *prompt;
```

DESCRIPTION

Getpass reads a password 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.

FILES

/dev/tty

SEE ALSO

crypt(3)

RESTRICTIONS

The return value points to static data whose content is overwritten by each call.

NAME

getpw - get name from UID

SYNOPSIS

```
getpw(uid, buf)
char *buf;
```

DESCRIPTION

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

getpwent, getpwuid, getpwnam, setpwent, endpwent - 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();
```

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.

```
/*
 * SCCSID: @(#)pwd.h     1.0  11/12/83
 */
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;
};
```

The fields pw_quota and pw_comment are unused; the others have meanings described in passwd(5).

Getpwent reads the next line (opening the file if necessary); setpwent rewinds the file; endpwent closes it.

Getpwuid and getpwnam search from the beginning until a matching uid or name is found (or until EOF is encountered).

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), passwd(5)

DIAGNOSTICS
Null pointer (0) returned on EOF or error.

RESTRICTIONS
All information is contained in a static area so it must be copied if it is to be saved.

GETS(3S)

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 a string into s from the standard input stream stdin. The string is terminated by a newline character, which is replaced in s by a null character. Gets returns its argument.

Fgets reads n-1 characters, or up to a newline character, whichever comes first, from the stream into the string s. The last character read into s is followed by a null character. Fgets returns its first argument.

SEE ALSO

puts(3), getc(3), scanf(3), fread(3), ferror(3)

DIAGNOSTICS

Gets and fgets return the constant pointer NULL upon end of file or error.

RESTRICTIONS

Gets deletes a newline, fgets keeps it, all in the name of backward compatibility.

NAME

hypot, cabs - euclidean distance

SYNOPSIS

```
#include <math.h>

double hypot(x, y)
double x, y;

double cabs(z)
struct { double x, y;} z;
```

DESCRIPTION

Hypot and cabs return

```
sqrt(x*x + y*y),
```

taking precautions against unwarranted overflows.

SEE ALSO

exp(3) for sqrt

J0(3M)

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;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause y0, y1, and yn to return a huge negative value and set errno to EDOM.

NAME

l3tol, ltol3 - convert between 3-byte integers and long integers

SYNOPSIS

```
l3tol(lp, cp, n)
long *lp;
char *cp;

ltol3(cp, lp, n)
char *cp;
long *lp;
```

DESCRIPTION

L3tol converts a list of n three-byte integers packed into a character string pointed to by cp into a list of long integers pointed to by lp.

Ltol3 performs the reverse conversion from long integers (lp) to three-byte integers (cp).

These functions are useful for file-system maintenance; disk addresses are three bytes long.

SEE ALSO

filsys(5)

MALLOC(3)

NAME

malloc, free, realloc, calloc - main 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;
```

DESCRIPTION

Malloc and free provide a simple general-purpose memory allocation package. Malloc returns a pointer to a block of at least size bytes beginning on a word boundary.

The argument to free is a pointer to a block previously allocated by malloc; this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by malloc is overrun or if some random number is handed to free.

Malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls sbrk (see break(2)) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to 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.

Realloc also works if ptr points to a block freed since the last call of malloc, realloc or calloc; thus sequences of free, malloc and realloc can exploit the search strategy of malloc to do storage compaction.

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for

storage of any type of object.

DIAGNOSTICS

Malloc, realloc and calloc return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. Malloc may be recompiled to check the arena very stringently on every transaction; see the source code.

RESTRICTIONS

When realloc returns 0, the block pointed to by ptr may be destroyed.

MKTEMP(3)

NAME

mktemp - make a unique file name

SYNOPSIS

```
char *mktemp(template)
char *template;
```

DESCRIPTION

Mktemp replaces template by a unique file name, and returns the address of the template. The template should look like a file name with six trailing X's, which will be replaced with the current process id and a unique letter.

SEE ALSO

getpid(2)

NAME

monitor - prepare execution profile

SYNOPSIS

```
monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)( ), (*highpc)( );
short buffer[ ];
```

DESCRIPTION

An executable program created by 'cc -p' automatically includes calls for monitor with default parameters; monitor needn't be called explicitly except to gain fine control over profiling.

Monitor is an 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. Monitor arranges to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. The lowest address sampled is that of lowpc and the highest is just below highpc. At most nfunc call counts can be kept; only calls of functions compiled with the profiling option -p of cc(1) are recorded. 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.

To profile the entire program, it is sufficient to use

```
extern etext();
...
monitor((int)2, etext, buf, bufsize, nfunc);
```

Etext lies just above all the program text, see end(3).

To stop execution monitoring and write the results on the file mon.out, use

```
monitor(0);
```

then prof(1) can be used to examine the results.

FILES

mon.out

SEE ALSO

prof(1), profil(2), cc(1)

NAME

itom, madd, msub, mult, mdiv, min, mout, pow, gcd, rpow - multiple precision integer arithmetic

SYNOPSIS

```
typedef struct { int len; short *val; } mint;

madd(a, b, c)
msub(a, b, c)
mult(a, b, c)
mdiv(a, b, q, r)
min(a)
mout(a)
pow(a, b, m, c)
gcd(a, b, c)
rpow(a, b, c)
msqrt(a, b, r)
mint *a, *b, *c, *m, *q, *r;

sdiv(a, n, q, r)
mint *a, *q;
short *r;

mint *itom(n)
```

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. After that space is managed automatically by the routines.

madd, msub, 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. rpow calculates a raised to the power b, while pow calculates this reduced modulo m. min andmout do decimal input and output.

The functions are obtained with the loader option -lmp.

DIAGNOSTICS

Illegal operations and running out of memory produce messages and core images.

NAME

nlist - get entries from name list

SYNOPSIS

```
#include <a.out.h>
nlist(filename, nl)
char *filename;
struct nlist nl[ ];
```

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See a.out(5) for the structure declaration.

This subroutine is useful for examining the system name list kept in the file /unix. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

NAME

perror, sys_errlist, sys_nerr - system error messages

SYNOPSIS

```
perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];
```

DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string s is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which 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)

NAME

pkopen, pkclose, pkread, pkwrite, pkfail - packet driver simulator

SYNOPSIS

```
char *pkopen(fd)

pkclose(ptr)
char *ptr;

pkread(ptr, buffer, count)
char *ptr, *buffer;

pkwrite(ptr, buffer, count)
char *ptr, *buffer;

pkfail()
```

DESCRIPTION

These routines are a user-level implementation of the full-duplex end-to-end communication protocol described in pk(4). If fd is a file descriptor open for reading and writing, pkopen carries out the initial synchronization and returns an identifying pointer. The pointer is used as the first parameter to pkread, pkwrite, and pkclose.

Pkread, pkwrite and pkclose behave analogously to read, write and close(2). However, a write of zero bytes is meaningful and will produce a corresponding read of zero bytes.

SEE ALSO

pk(4), pkon(2)

DIAGNOSTICS

Pkfail is called upon persistent breakdown of communication. Pkfail must be supplied by the user.
Pkopen returns a null (0) pointer if packet protocol can not be established.
Pkread returns -1 on end of file, 0 in correspondence with a 0-length write.

RESTRICTIONS

The Packet Driver is not supported by the ULTRIX-11 operating system.

This simulation of pk(4) leaves something to be desired in needing special read and write routines, and in not being inheritable across calls of exec(2). Its prime use is on systems that lack pk.
These functions use alarm(2); simultaneous use of alarm for other puposes may cause trouble.

NAME

plot: openpl et al. - 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 newlines.

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(1) filters
-lt300 GSI 300 terminal
-lt300s GSI 300S terminal
-lt450 DASI 450 terminal
-lt4014 Tektronix 4014 terminal

SEE ALSO

plot(5), plot(1), graph(1)

NAME

popen, pclose - initiate I/O to/from 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. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

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, and a type "w" as an output filter.

SEE ALSO

pipe(2), fopen(3), fclose(3), system(3), wait(2)

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the Shell cannot be accessed.

Pclose returns -1 if stream is not associated with a 'popened' command.

RESTRICTIONS

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, e.g. with fflush, see fclose(3).

NAME

printf, fprintf, sprintf - formatted output conversion

SYNOPSIS

```
#include <stdio.h>

printf(format [, arg ] ... )
char *format;

fprintf(stream, format [, arg ] ... )
FILE *stream;
char *format;

sprintf(s, format [, arg ] ... )
char *s, format;
```

DESCRIPTION

Printf places output on the standard output stream stdout. Fprintf places output on the named output stream. Sprintf places 'output' in the string s, followed by the character '\0'.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument 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 the next successive arg printf.

Each conversion specification is introduced by the character %. Following the %, there may be

- an optional minus sign '-' which specifies left adjustment of the converted value in the indicated field;
- an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
- an optional period '.' which serves to separate the field width from the next digit string;
- an optional digit string specifying a precision which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

- the character l specifying that a following d, o, x, or u corresponds to a long integer arg. (A capitalized conversion code accomplishes the same thing.)

- a character which indicates the type of conversion to be applied.

A field width or precision may be '*' instead of a digit string. In this case an integer arg supplies the field width or precision.

The conversion characters and their meanings are

dox The integer arg is converted to decimal, octal, or hexadecimal notation respectively.

f The float or double arg is converted to decimal notation in the style '[-]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.

e The float or double arg is converted in the style '[-]d.ddde+dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.

g The float or double arg is printed in style d, in style f, or in style e, whichever gives full precision in minimum space.

c The character arg is printed. Null characters are ignored.

s Arg is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.

u The unsigned integer arg is converted to decimal and printed (the result will be in the range 0 to 65535).

% Print a '%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by printf are printed by putc(3).

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 %d,  %02d:%02d", weekday, month, day,
     hour, min);
```

To print pi to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO
putc(3), scanf(3), ecvt(3)

RESTRICTIONS
Very wide fields (>128 characters) fail.

NAME

putc, putchar, fputc, putw - put character or word on a stream

SYNOPSIS

```
#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;
```

DESCRIPTION

Putc appends the character c to the named output stream. It returns the character written.

Putchar(c) is defined as putc(c, stdout).

Fputc behaves like putc, but is a genuine function rather than a macro. It may be used to save on object text.

Putw appends word (i.e. int) w to the output stream. It returns the word written. Putw neither assumes nor causes special alignment in the file.

The standard stream stdout is normally buffered if and only if the output does not refer to a terminal; this default may be changed by setbuf(3). The standard stream stderr is by default unbuffered unconditionally, but use of freopen (see fopen(3)) will cause it to become buffered; setbuf, again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. Fflush (see fclose(3)) may be used to force the block out early.

SEE ALSO

fopen(3), fclose(3), getc(3), puts(3), printf(3), fread(3)

DIAGNOSTICS

These functions return the constant EOF upon error. Since this is a good integer, ferror(3) should be used to detect putw errors.

PUTC(3S)

RESTRICTIONS

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular 'putc(c, *f++);' doesn't work sensibly.

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 copies the null-terminated string s to the standard output stream stdout and appends a newline character.

Fputs copies the null-terminated string s to the named output stream.

Neither routine copies the terminal null character.

SEE ALSO

fopen(3), gets(3), putc(3), printf(3), ferror(3)
fread(3) for fwrite

RESTRICTIONS

Puts appends a newline, fputs does not, all in the name of backward compatibility.

QSORT(3)

NAME

qsort - quicker sort

SYNOPSIS

```
qsort(base, nel, width, compar)
char *base;
int (*compar)( );
```

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO

sort(1)

NAME

rand, srand - random number generator

SYNOPSIS

```
srand(seed)
int seed;

rand( )
```

DESCRIPTION

Rand uses a multiplicative congruential random number generator with period 232 to return successive pseudo-random numbers in the range from 0 to 215-1.

The generator is reinitialized by calling srand with 1 as argument. It can be set to a random starting point by calling srand with whatever you like as argument.

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 control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.

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, and a conversion character.

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 *. 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.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. 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.

o an octal integer is expected; the corresponding argument should be a integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

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. The input field is terminated by a space character or a newline.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

e f 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 an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.

[indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters d, o and x may be capitalized or preceeded by l to indicate that a pointer to long rather than to int is in the argument list. Similarly, the conversion characters e or f may be capitalized or preceded by l to indicate a pointer to double rather than to float. The

conversion characters d, o and x may be preceeded by h to indicate a pointer to short rather than to int.

The scanf functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant EOF is returned upon end of input; note that 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.

For example, the call

```
int i; float x; char name[50];
scanf( "%d%f%s", &i, &x, name);
```

with the input line

```
25   54.32E-1  thompson
```

will assign to i the value 25, x the value 5.432, and name will contain 'thompson\0'. Or,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to i, 789.0 to x, skip '0123', and place the string '56\0' in name. The next call to getchar will return 'a'.

SEE ALSO

atof(3), getc(3), printf(3)

DIAGNOSTICS

The scanf functions return EOF on end of input, and a short count for missing or illegal data items.

RESTRICTIONS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

setbuf - assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;
```

DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array buf to be used instead of an automatically allocated buffer. If buf is the constant pointer NULL, input/output will be completely unbuffered.

A manifest constant BUFSIZ tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from malloc(3) upon the first getc or putc(3) on the file, except that output streams directed to terminals, and the standard error stream stderr are normally not buffered.

SEE ALSO

fopen(3), getc(3), putc(3), malloc(3)

NAME

setjmp, longjmp - non-local goto

SYNOPSIS

```
#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;
```

DESCRIPTION

These routines 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. It returns value 0.

Longjmp restores the environment saved by the last call of setjmp. It 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, which must not itself have returned in the interim. All accessible data have values as of the time longjmp was called.

SEE ALSO

signal(2)

NAME

sigset, signal, sighold, sigignore, sigrelse, sigpause - manage signals

SYNOPSIS

```
#include <signal.h>
void action();
int action(); (if void is not supported)
int sig;

sigset(sig, action)
signal(sig, action)

sighold(sig)
sigignore(sig)
sigrelse(sig)

sigpause(sig)

cc ... -ljobs
```

DESCRIPTION

This is a package of signal management functions to manage the signals as described in sigsys(2J). These functions are not available in most versions of UNIX, and should not be used when the mechanisms of signal(2) would suffice, as they would then impair portability. These functions are contained in the jobs library, obtained by specifying the loader option -ljobs.

Sigset is used to provide a default signal handler for signal sig. This function is remembered across subsequent calls to the other functions, and need not be specified again. After sigset instances of sig will cause an interrupt to be taken at func, with the signal then held so that recursive trapping due to the signal will not occur. During normal return from func, the routines arrange for the signal action to be restored so that subsequent signals will also trap to func. If a non-local exit is to be taken, then sigrelse must be called to un-hold the signal action, restoring the original catch. Func may also be specified as SIG_DFL, SIG_IGN or SIG_HOLD, as described in sigsys(2J). The value specified on the previous call to sigset is returned; if sigset has never been called, then the default action inherited from the system is returned.

Signal is like sigset, but the signal will not be held when the action routine is called; rather it will have reverted to SIG_DFL. This is generally unsafe, but is included for backwards compatibility to the old signal mechanism. It should not be used.

Sighold and sigrelse may be used to block off sig in a piece of code where it cannot be tolerated. After sigrelse the catch initially set with sigset will be restored.

Sigignore can be used to temporarily set the action for sig to ignore the signal. If the signal had been held before the call to sigignore, any pending instance of the signal will be discarded.

Sigpause may be used by a routine which wishes to check for some condition produced at interrupt level by the sig signal, and then to pause waiting for the condition to arise with the catch of the signal enabled. In correct usage it must be preceded by an instance of sighold to block the signal. Sigpause is like pause in that it will return after any signal is processed. The usual thing to do then is to reenable the hold with sighold, check the condition again, and sigpause again if the condition has not arisen.

SEE ALSO

signal(2), sigsys(2J), intro(3J), tty(4)

RESTRICTIONS

Sighold and sigrelse do not nest; the first sigrelse restores the default catch.

These functions store information in data space. You thus must call sigsys(2J) rather than any of sigset or signal after a vfork(2) in the child which is to then exec.

NAME

sin, cos, tan, asin, acos, atan, atan2 - trigonometric functions

SYNOPSIS

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(x, y)
double x, y;
```

DESCRIPTION

Sin, cos and tan return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

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 returns the arc tangent of x/y in the range -pi to pi.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause asin and acos to return value 0; errno is set to EDOM. The value of tan at its singular points is a huge number, and errno is set to ERANGE.

RESTRICTIONS

The value of tan for arguments greater than about 2**31 is garbage.

NAME

sinh, cosh, tanh - hyperbolic functions

SYNOPSIS

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;
```

DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and cosh return a huge value of appropriate sign when the correct value would overflow.

NAME

sleep - suspend execution for interval

SYNOPSIS

```
sleep(seconds)
unsigned seconds;
```

DESCRIPTION

The current process is suspended 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 an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an alarm clock signal and pausing until it occurs. The previous state of this signal is saved and restored. If the sleep time exceeds the time to the alarm signal, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

alarm(2), pause(2)

NAME

stdio - standard buffered input/output package

SYNOPSIS

```
#include <stdio.h>
FILE *stdin;
FILE *stdout;
FILE *stderr;
```

DESCRIPTION

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros getc and putc(3) handle characters quickly. The higher level routines gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite all 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(3) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

```
stdin     standard input file
stdout    standard output file
stderr    standard error file
```

A constant 'pointer' NULL (0) designates no stream at all.

An integer constant EOF (-1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include 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.

SEE ALSO

open(2), close(2), read(2), write(2)

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.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex - string operations

SYNOPSIS

```
char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s;
```

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 copies at most n characters. Both return 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 looks at at most n characters.

Strcpy copies string s2 to s1, stopping after the null character has been moved. Strncpy copies exactly n characters, truncating or null-padding s2; the target may not be null-terminated if the length of s2 is n or more. Both return s1.

STRING(3)

Strlen returns the number of non-null characters in s.

Index (rindex) returns a pointer to the first (last) occurrence of character c in string s, or zero if c does not occur in the string.

RESTRICTIONS

Strcmp uses native character comparison, which is signed on PDP11's, unsigned on other machines.

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 PDP11's and other machines. Nbytes should be even.

SYSTEM(3)

NAME

system - issue a shell command

SYNOPSIS

```
system(string)
char *string;
```

DESCRIPTION

System causes the string to be given to sh(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO

popen(3), exec(2), wait(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

NAME

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(3) for a higher level package.

Tgetent extracts the entry for terminal name into the buffer at bp. 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 can't write the file /etc/termcap.

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.

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 that programs using termcap should in general turn off XTABS anyway since some terminals use control 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 output speed of the terminal as encoded by stty (2). The external variable PC should contain a pad character to be used (from the pc capability) if a null (^@) is inappropriate.

FILES

/usr/lib/libtermlib.a	-ltermlib library
/etc/termcap	data base

SEE ALSO

ex(1), curses(3), termcap(5)

AUTHOR

William Joy

RESTRICTIONS

TTYNAME(3)

NAME

ttyname, isatty, ttyslot - find name of a terminal

SYNOPSIS

```
char *ttyname(fildes)

isatty(fildes)

ttyslot()
```

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor fildes.

Isatty returns 1 if fildes is associated with a terminal device, 0 otherwise.

Ttyslot returns the number of the entry in the ttys(5) file for the control terminal of the current process.

FILES

/dev/*
/etc/ttys

SEE ALSO

ioctl(2), ttys(5)

DIAGNOSTICS

Ttyname returns a null pointer (0) if fildes does not describe a terminal device in directory '/dev'.

Ttyslot returns 0 if '/etc/ttys' is inaccessible or if it cannot determine the control terminal.

RESTRICTIONS

The return value points to static data whose content is overwritten by each call.

NAME

ungetc - push character back into input stream

SYNOPSIS

```
#include <stdio.h>

ungetc(c, stream)
FILE *stream;
```

DESCRIPTION

Ungetc pushes the character c back on an input stream. That character will be returned by the next getc call on that stream. Ungetc returns c.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3) erases all memory of pushed back characters.

SEE ALSO

getc(3), setbuf(3), fseek(3)

DIAGNOSTICS

Ungetc returns EOF if it can't push a character back.

CAT(4)

NAME

cat - phototypesetter interface

DESCRIPTION

Cat provides the interface to a Graphic Systems C/A/T phototypesetter, via DR11-C general device interface. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding will not be described here.

Only one process may have this file open at a time. It is write-only.

FILES

/dev/cat

SEE ALSO

troff(1)
Phototypesetter interface specification
ULTRIX-11 System Management Guide, Section 2.6 and Appendix A

RESTRICTIONS

The C/A/T interface is only partially supported by ULTRIX-11, i.e., data written to the /dev/cat file will be presented at the output cable of the DR11-C.

NAME

dh, dz - DH11/DHU11/DHV11 and DZ11/DZV11/DZQ11 asynchronous multiplexers dhdm - DM11-BB (modem controller for DH11)

DESCRIPTION

Each line attached to a DH11, DZ11, DZV11, or DZQ11 communications multiplexer behaves as described in tty (4). Input and output for each line may independently be set to run at any of 16 speeds; see tty (4) for encoding. For DZ11, DZV11 and DZQ11 lines, the output speed always equals the input speed. The 200 speed and the two externally clocked speeds (exta, extb) are missing on the DZ11, DZV11 and DZQ11. The 50 speed and extb are not allowed on the DHU11 and DHV11.

The DH11 and DHU11 have 16 lines, the DZ11 and DHV11 have 8 lines, and the DZV11 and DZQ11 have four lines. The DHV11, DZV11 and DZQ11 are used only on Q bus processors.

FILES

/dev/tty##

SEE ALSO

tty(4), getty(8)
ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

NAME

dn - DN-11 ACU interface

DESCRIPTION

The dn# files are write-only. The permissible codes are:

0-9	dial 0-9
:	dial *
;	dial #
-	4 second delay for second dial tone
<	end-of-number

The entire telephone number must be presented in a single write system call.

It is recommended that an end-of-number code be given even though not all ACU's actually require it.

FILES

/dev/dn0	connected to 801 with dp0
/dev/dn1	not currently connected
/dev/dn2	not currently connected

SEE ALSO

du(4)

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A

ULTRIX-11 Software Technical Description

RESTRICTIONS

The DN-11 is obsolete and is not supported by ULTRIX-11.

NAME

du, dp - DU-11 201 data-phone interface

DESCRIPTION

The dp0 file is a 201 data-phone interface. Read and write calls to dp0 are limited to a maximum of 512 bytes. Each write call is sent as a single record. Seven bits from each byte are written along with an eighth odd parity bit. The sync must be user supplied. Each read call returns characters received from a single record. Seven bits are returned unaltered; the eighth bit is set if the byte was not received in odd parity. A 10 second time out is set and a zero-byte record is returned if nothing is received in that time.

The unibus interrupt request priority level for the DU-11 must be set to BR 6.

FILES

/dev/dp0
/dev/du#

SEE ALSO

dn(4)
ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

The DU-11 is obsolete and is not supported by ULTRIX-11.
The name dp0 is a historical dreg.

NAME

dh, dz - DH11/DHU11/DHV11 and DZ11/DZV11/DZQ11 asynchronous multiplexers dhdm - DM11-BB (modem controller for DH11)

DESCRIPTION

Each line attached to a DH11, DZ11, DZV11, or DZQ11 communications multiplexer behaves as described in tty (4). Input and output for each line may independently be set to run at any of 16 speeds; see tty (4) for encoding. For DZ11, DZV11 and DZQ11 lines, the output speed always equals the input speed. The 200 speed and the two externally clocked speeds (exta, extb) are missing on the DZ11, DZV11 and DZQ11. The 50 speed and extb are not allowed on the DHU11 and DHV11.

The DH11 and DHU11 have 16 lines, the DZ11 and DHV11 have 8 lines, and the DZV11 and DZQ11 have four lines. The DHV11, DZV11 and DZQ11 are used only on Q bus processors.

FILES

/dev/tty##

SEE ALSO

tty(4), getty(8)
ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

NAME

hk - RK611/RK06, RK07 moving-head disk

DESCRIPTION

The RK611/RK711 controller supports up to eight RK06/RK07 disk drives.

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for information about logical disk names and Appendix D for disk partition sizes.

The hk files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records.

A 'raw' interface provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and 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 the buffer must begin on a word boundary.

FILES

/dev/hk##, /dev/rhk##

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

hp - First RH11/RH70 massbus disk controller
hm - Second RH11/RH70 massbus disk controller
hj - Third RH11/RH70 massbus disk controller

DESCRIPTION

The HP disk driver supports a heterogeneous mix of up to eight RM02/3/5, RP04/5/6, and ML11 disks on the first RH11/RH70 controller, normally addressed at 0776700. The HM driver supports an identical mix of disks on the second and third RH11/RH70 controllers, addressed at 0776300 and 0776400 respectively. The RM03 and RM05 disks may not be used with the RH11 controller because their transfer rates exceed the bandwidth of the unibus.

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for information about logical disk names and Appendix D for disk partition sizes.

The ML11 may be used for the swap device or perhaps mounted on /tmp. The ML11 has switch-selectable transfer rates of 0.25 mb, 0.5 mb, 1.0 mb, and 2.0 mb per second. The following transfer rate restrictions apply:

0.25 mb	all CPU's
0.5 mb	all CPU's
1.0 mb	PDP 11/70 with RH70 only
2.0 mb	NO PDP11 CPU's

The hp and hm files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. A 'raw' interface provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and 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 the buffer must begin on a word boundary.

FILES

/dev/hp##, /dev/rhp##
/dev/hm##, /dev/rhm##

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

HP(4)

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

hs - RH11/RH70 - RS03, RS04 fixed-head disk

DESCRIPTION

The files rs0 ... rs7 refer to RS03 disk drives 0 through 7. The files hs0 ... hs7 refer to RS04 disk drives 0 through 7. The RS03 drives are each 1024 blocks long and the RS04 drives are 2048 blocks long.

The hs and rs files access the disk via 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 results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw HS files begin with rhs. The same minor device considerations hold for the raw interface as for the normal interface. In raw I/O the buffer must begin on a word boundary.

FILES

/dev/rx#, /dev/rrs#
/dev/hs#, /dev/rhs#

SEE ALSO

ULTRIX-11 Software Technical Description

RESTRICTIONS

This device is obsolete and, although a driver is provided, is not supported by ULTRIX-11.

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

HT(4)

NAME

ht - RH11/RH70 - TM02/3 magtape controller

DESCRIPTION

Each TM02/3 magtape controller supports up to eight TU16/TE16/TU77 tape drives. Eight TM02/3 controllers may be connected to a single RH11 or RH70 massbus controller.

When opened for reading or writing, the tape is not rewound. When closed, it is rewound (unless the 0200 bit is on, see below). If the tape was open for writing, a double end-of-file is written. If the tape is not to be rewound the tape is backspaced to just between the two tapemarks.

A standard tape consists of a series of 512 byte records terminated by a double end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The last octal digit of the minor device number selects the drive. The middle digit selects a controller. The initial digit is even to select 1600 BPI, odd to select 800 BPI. If the 0200 bit is on (initial digit 2 or 3), the tape is not rewound on close. Note that the minor device number has no necessary connection with the file name.

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for more information about logical device names.

The mt files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated files may be named rmt0, ..., rmt7, but the same minor-device considerations as for the regular files still apply.

Each read or write 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; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. A zero count is returned when a tape mark is read; another read will fetch the first record of the next tape file.

FILES

/dev/mt#, /dev/rmt#, /dev/nrmt#
/dev/ht#, /dev/rht#, /dev/nrht#

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

The magtape system is supposed to be able to take 64 drives. Such addressing has never been tried.

Taking a drive off line, or running off the end of tape, while writing have been known to hang the system.

If any non-data error is encountered, it refuses to do anything more until closed. In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark explicitly.

NAME

hx - RX211/RX02 floppy disk

DESCRIPTION

HX? refers to an entire disk as a single sequentially-addressed file. The physical disk sector size is 128 bytes for single density and 256 bytes for double density, the logical block size is 512 bytes. Each diskette has 500 logical blocks, single density and 1001 logical blocks, double density. The minor device numbers have the following significance:

name	minor device	unit	density	format
hx0	0	0	single	interleaved
hx1	1	1	single	interleaved
hx2	2	0	double	interleaved
hx3	3	1	double	interleaved
hx4	4	0	single	physical
hx5	5	1	single	physical
hx6	6	0	double	physical
hx7	7	1	double	physical
hx8	8	0	single	extended
hx9	9	1	single	extended

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for more information about logical disk names and Appendix D for disk sizes.

The hx files discussed above access the disk via 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 results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw HX files begin with rhx and end with a number which selects the same disk as the corresponding hx file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise seek calls should specify a multiple of 512 bytes.

A logical 512 byte block is made up of four physical sectors on the single density devices, and two physical sectors on the double density devices. The four 'physical' device names are a straight mapping from the logical block number to the physical sectors, thus logical block 0 maps to track 0, sectors 1-2 for double density, and track 0, sectors 1-4 for single density. This is not very efficient, since the

controller spends most of it's time waiting for the disk to get into position. The four 'interleaved' devices map logical blocks to every other sector in the track. In addition, there is a six sector slew between tracks. Thus, logical block 0 would be track 1, sectors 1, 3, 5 and 7 for single density, and sectors 1 and 3 for double density. Note that the 'interleaved' devices start with track 1, go to track 76, and then to track 0, where as the 'physical' devices start with track 0.

There are a total of 2002 sectors on a floppy, which for single density is a total of 500.5 512 byte blocks. The 'interleaved' devices ignore the last half block (track 0, sectors 24 and 26). This last half block can be accessed by using the 'extended' devices, which have a size of 501 blocks. When writing block 501, only the first 256 bytes will be written, the rest is discarded. When reading block 501, the last 256 bytes are guaranteed to be nulls. These devices are intended for floppy image copies only, and should NOT be used for normal filesystems.

FILES

/dev/hx#, /dev/rhx#

SEE ALSO

rx2fmt(1)
ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

lp - line printer

DESCRIPTION

Lp provides the interface to any of the standard Digital Equipment Corporation line printers, on an LP-11 parallel interface. Bytes written are printed.

The driver has several internal parameters for adapting to the characteristics of various printers. These parameters are settable, via ioctl(2) call, using the lpset(1) command.

The FFCLOSE parameter causes one page eject on close. The default is no page eject on close, because the line printer spooler does a page eject after each file is printed. No page eject is done on open.

The CAP parameter determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

{	↤
}	↦
`	↥
|	↕
~	△

The driver correctly interprets carriage returns, backspaces, tabs, and form-feeds. A new-line that extends over the end of a page is turned into a form-feed. The default line length is 132 characters, indent is 0 characters and lines per page is 66. These parameters are also settable by lpset(1). Lines longer than the line length minus the indent are truncated.

FILES

/dev/lp

SEE ALSO

lpr(1), lpset(1)

RESTRICTIONS

Only a single LP11 line printer controller is supported.

NAME

mem, kmem - core memory

DESCRIPTION

Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system. Kmem is the same as mem except that kernel virtual memory rather than physical memory is accessed.

Byte addresses are interpreted as memory addresses. References to non-existent locations return errors.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

On PDP11's, the I/O page begins at location 0160000 of kmem and per-process data for the current process begins at 0140000.

FILES

/dev/mem, /dev/kmem

RESTRICTIONS

On PDP11's, memory files are accessed one byte at a time, an inappropriate method for some device registers.

NAME

ml - RH11/RH70 - ML11 solid state disk

DESCRIPTION

The ML driver supports up to eight ML11 disk units on a separate RH11 or RH70 massbus disk controller. If the ML11 is mixed with other disks, the HP or HM driver must be used.

ml? refers to an entire ML11 unit as a single sequentially addressed file. The size of each ML11 unit depends on the number of array modules installed. There are 512 512-byte blocks per array module, for a maximum size of 8192 blocks per unit.

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for more information about logical disk names and Appendix D for disk sizes.

The ML11 may be used for the swap device or perhaps mounted on /tmp. The ML11 has switch-selectable transfer rates of 0.25 mb, 0.5 mb, 1.0 mb, and 2.0 mb per second. The following transfer rate restrictions apply:

0.25 mb	all CPU's
0.5 mb	all CPU's
1.0 mb	PDP 11/70 with RH70 only
2.0 mb	NO PDP11 CPU's

The ml files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. A 'raw' interface provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and 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 the buffer must begin on a word boundary.

FILES

/dev/ml#, /dev/rml#

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

newtty - summary of the 'new' tty driver

USAGE

stty new

stty new crt

DESCRIPTION

This is a summary of the new tty driver, described completely, with the old terminal driver, in tty(4). The new driver is largely compatible with the old but provides additional functionality for job control.

CRTs and printing terminals.

The new terminal driver is normally set differently on CRTs and on printing terminals. On CRTs at speeds of 1200 baud or greater it normally erases input characters physically with backspace-space-backspace when they are erased logically; at speed under 1200 baud this is often unreasonably slow, so the cursor is normally merely moved to the left. This is the behavior after a "stty new crt"; to have the tty driver always erase the characters use "stty new crt crterase crtkill"; to have the characters remain even at 1200 baud or greater use "stty new crt -crterase -crtkill".

On printing terminals the command "stty new prterase" should be given. Logically erased characters are then echoed printed backwards between a '\' and an '/' character.

Other terminal modes are possible, but less commonly used; see tty(4) and stty(1) for details.

Input editing and output control.

When preparing input the character # (normally changed to ^H using stty(1)) erases the last input character, ^W the last input word, and the character @ (often changed to ^U) erases the entire current input line. A ^R character causes the pending input to be retyped. Lines are terminated by a return or a newline; a ^D at the beginning of a line generates an end-of-file.

Control characters echo as ^x when typed, for some x; the delete character is represented as ^?.

The character ^V may be typed before any character so that it may be entered without its special effect. For backwards compatibility with the old tty driver the character '\' prevents the special meaning of the character and line erase characters, much as ^V does.

NEWTTY(4)

Output is suspended when a ^S character is typed and resumed when a ^Q character is typed. Output is discarded after a ^O character is typed until another ^O is type, more input arrives, or the condition is cleared by a program (such as the shell just before it prints a prompt.)

Signals.

A non-interactive program is interrupted by a ^? (delete); this character is often reset to ^C using stty(1). A quit ^\ character causes programs to terminate like ^? does, but also causes a core image file to be created which can then be examined with a debugger. This is often used to stop runaway processes. Interactive programs often catch interrupts and return to their command loop; only the most well debugged programs catch quits.

Programs may be stopped by hitting ^Z, which returns control to the shell. They may then be resumed using the job control mechanisms of the shell. The character ^Y is like ^Z but takes effect when read rather then when typed; it is much less frequently used.

See tty(4) for a more complete description of the new terminal driver.

SEE ALSO

csh(1), stty(1), tty(4)

NAME

null - data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

tty - general terminal interface

DESCRIPTION

This section describes both a particular special file, and the general nature of the terminal interface.

The file /dev/tty is, in each process, a synonym for the control terminal associated with that process. It is useful for programs 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 a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

As for terminals in general: all of the low-speed asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of the interface.

When a terminal file is opened, it causes the process to wait until a connection is established, unless the terminal line is set for local operation, see ttys (5). In practice user's programs seldom open these files; they are opened by init and become a user's input and output file. The very first terminal file open in a process becomes the control terminal for that process. The control terminal plays a special role in handling quit or interrupt signals, as discussed below. The control terminal is inherited by a child process during a fork, even if the control terminal is closed. The set of processes that thus share a control terminal is called a process group; all members of a process group receive certain signals together, see ^C below and kill(2).

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 choked, 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. When the input limit is reached all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. 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. There are special modes, discussed below, that permit the program to read each character as typed without waiting for a full line.

During input, erase and kill processing is normally done. By default, the character 'DELETE' erases the last character typed, except that it will not erase beyond the beginning of a line or an EOT. By default, the character 'Control-U' kills the entire line up to the point where it was typed, but not beyond an EOT. Both these characters operate on a keystroke basis independently of any backspacing or tabbing that may have been done. Either 'Control-U' or 'DELETE' may be entered literally by preceding it by '\'; the erase or kill character remains, but the '\' disappears. These two characters may be changed to others.

When desired, all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by '\'. In addition, the following escape sequences can be generated on output and accepted on input:

for	use
`	\'
|	\!
~	\^
{	\(
}	\)

Certain ASCII control characters have special meaning. These characters are not passed to a reading program except in raw mode where they lose their special character. Also, it is possible to change these characters from the default; see below.

EOT (Control-D) may be used to generate an end of file from a terminal. When an EOT is received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOT is discarded. Thus if there are no characters waiting, which is to say the EOT occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file indication.

^C (Control-C) is not passed to a program but generates an interrupt signal which is sent to all processes with the associated control 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 signal(2).

FS (Control-\) generates the quit signal. Its treatment

OLDTTY(4)

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 will be generated.

DC3 (Control-S) delays all printing on the terminal until something is typed in.

DC1 (Control-Q) restarts printing after DC3 without generating any input to a program.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a hangup signal is sent to all processes with the terminal as control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file on their input can terminate appropriately when hung up on.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. When 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. Even parity is always generated on output. The EOT character is not transmitted (except in raw mode) to prevent terminals that respond to it from hanging up.

Several ioctl(2) calls apply to terminals. Most of them use the following structure, defined in <sgtty.h>:

```
struct sgttyb {
     char sg_ispeed;
     char sg_ospeed;
     char sg_erase;
     char sg_kill;
     int  sg_flags;
};
```

The sg_ispeed and sg_ospeed fields describe the input and output speeds of the device according to the following table, which corresponds to the DEC DH-11 interface. If other hardware is used, impossible speed changes are ignored. Symbolic values in the table are as defined in <sgtty.h>.

```
B0      0    (hang up dataphone)
B50     1    50 baud
```

B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A
EXTB	15	External B

In the current configuration, only 110, 150, 300 and 1200 baud are really supported on dial-up lines. Code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program. The half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied; full-duplex 212 datasets work fine.

The sg erase and sg kill fields of the argument structure specify the erase and kill characters respectively. (Defaults are DELETE and Control-U.)

The sg flags field of the argument structure contains several bits that determine the system's treatment of the terminal:

```
ALLDELAY 0177400 Delay algorithm selection
BSDELAY  0100000 Select backspace delays (not implemented):
BS0      0
BS1      0100000
VTDELAY  0040000 Select form-feed and vertical-tab delays:
FF0      0
FF1      0100000
CRDELAY  0030000 Select carriage-return delays:
CR0      0
CR1      0010000
CR2      0020000
CR3      0030000
TBDELAY  0006000 Select tab delays:
TAB0     0
TAB1     0001000
TAB2     0004000
XTABS    0006000
NLDELAY  0001400 Select new-line delays:
NL0      0
NL1      0000400
NL2      0001000
NL3      0001400
```

OLDTTY(4)

EVENP	0000200	Even parity allowed on input (most terminals)
ODDP	0000100	Odd parity allowed on input
RAW	0000040	Raw mode: wake up on all characters, 8-bit interface
CRMOD	0000020	Map CR into LF; echo LF or CR as CR-LF
ECHO	0000010	Echo (full duplex)
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Return each character as soon as typed
TANDEM	0000001	Automatic flow control

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.

Backspace delays are currently ignored but might be used for Terminet 300's.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is unimplemented and is 0.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

Characters with the wrong parity, as determined by bits 200 and 100, are ignored.

In raw mode, every character is passed immediately to the program without waiting until a full line has been typed. No erase or kill processing is done; the end-of-file indicator (EOT), the interrupt character (Control-C) and the quit character (FS) are not treated specially. There are no delays and no echoing, and no replacement of one character for another; characters are a full 8 bits for both input and output (parity is up to the program).

Mode 020 causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (for terminals with a new-line function).

CBREAK is a sort of half-cooked (rare?) mode. Programs can read each character as soon as typed, instead of waiting for a full line, but quit and interrupt work, and output delays, case-translation, CRMOD, XTABS, ECHO, and parity work normally. On the other hand there is no erase or kill, and no special treatment of \ or EOT.

TANDEM mode causes the system to produce a stop character (default DC3) whenever the input queue is in danger of overflowing, and a start character (default DC1) when the input queue has drained sufficiently. It is useful for flow control when the 'terminal' is actually another machine that obeys the conventions.

Several ioctl calls have the form:

```
#include <sgtty.h>

ioctl(fildes, code, arg)
struct sgttyb *arg;
```

The applicable codes are:

TIOCGETP
: Fetch the parameters associated with the terminal, and store in the pointed-to structure.

TIOCSETP
: Set the parameters according to the pointed-to structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOCGETD
: Fetch the line discipline number associated with the terminal, and store in the first character of the pointed-to structure.

TIOCSETD
: Set the line discipline number according to the the first character of the pointer-to structure.

TIOCSETN
: Set the parameters but do not delay or flush input. Switching out of RAW or CBREAK mode may cause some garbage input.

TIOCETQ
: Set the flags field of the pointed-to structure equal to the number of characters waiting on the RAW Q or the CANONICAL Q, which ever is greater.

OLDTTY(4)

With the following codes the arg is ignored.

TIOCEXCL
Set "exclusive-use" mode: no further opens are permitted until the file has been closed.

TIOCNXCL
Turn off "exclusive-use" mode.

TIOCHPCL
When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with an ACU used to place outgoing calls.

TIOCFLUSH
All characters waiting in input or output queues are flushed.

TIOCSDTR
Causes the Data Terminal Ready (DTR) signal to be asserted on the communications port associated with fildes.

TIOCCDTR
Removes the DTR signal from the communications port.

TIOCSBRK
Sets a break condition on the communications port associated with fildes.

TIOCCBRK
Removes the break condition from the communications port.

The following codes affect characters that are special to the terminal interface. The argument is a pointer to the following structure, defined in <sgtty.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 default values for these characters are Control-C, FS, DC1, DC3, EOT, and -1. A character value of -1 eliminates the effect of that character. The t_brkc character, by default -1, acts like a new-line in that it terminates a 'line,' is echoed, and is passed to the program. The 'stop'

and 'start' characters may be the same, to produce a toggle effect. It is probably counterproductive to make other special characters (including erase an kill) identical.

The calls are:

TIOCSETC
Change the various special characters to those given in the structure.

TIOCGETC
Fetch the special characters and store in the pointed to structure.

The following codes affect erase/kill processing modes for the terminal.
These ioctl's accept and return integer values only.

The calls are:

TIOCSETT
Set the erase/kill processing flags to the value in the pointed to integer.

TIOCGETT
Fetch the erase/kill flags and store in the pointed to integer.

The valid values for erase/kill processing flag are:

```
LCRTBS  000001 Backspace on erase, don't echo erase character.
LCRTERA 000002 Use backspace-space-backspace to rubout character.
LPRTERA 000004 Printing terminal erase mode.
LCRTKIL 000010 Use LCRTERA method to kill the line.
LCTLECH 000020 Echo input control chars as ^X, delete as ^?.
LCRTSLO 000040 Low speed (< 1200 baud). Use newline for kill.
```

FILES
/dev/tty
/dev/tty##
/dev/console

SEE ALSO
getty(8), stty (1), signal(2), ioctl(2), newtty(4), init(8)
ULTRIX-11 System Management Guide, Sections 1.4.5 and 2.6.3

OLDTTY(4)

RESTRICTIONS

Half-duplex terminals are not supported.

The terminal handler has clearly entered the race for ever-greater complexity and generality. It's still not complex and general enough for TENEX fans.

NAME

pk - packet driver

DESCRIPTION

The packet driver implements a full-duplex end-to-end flow control strategy for machine-to-machine communication. Packet driver protocol is established by calling pkon(2) with a character device file descriptor and a desired packet size in bytes. The packet size must be a power of 2, 32<=size<=4096. The file descriptor must represent an 8-bit data path. This is normally obtained by setting the device in raw mode (see ioctl(2)).

The actual packet size, which may be smaller than the desired packet size, is arrived at by negotiation with the packet driver at the remote end of the data link.

The packet driver maintains two data areas for incoming and outgoing packets. The output area is needed to implement retransmission on errors, and arriving packets are queued in the input area. Data arriving for a file not open for reading is discarded. Initially the size of both areas is set to two packets.

It is not necessary that reads and writes be multiples of the packet size although there is less system overhead if they are. Read operations return the maximum amount of data available from the input area up to the number of bytes specified in the system call. The buffer sizes in write operations are not normally transmitted across the link. However, writes of zero length are treated specially and are reflected at the remote end as a zero-length read. This facilitates marking the serial byte stream, usually for delimiting files.

When one side of a packet driver link is shut down by close(2)or pkoff (see pkon(2)), read(2) on the other side will return 0, and write on the other side will raise a SIGPIPE signal.

RESTRICTIONS

The pkon(2) system call does not exist. The packet driver is not supported by ULTRIX-11.

SEE ALSO

pkon(2), pkopen(3)
ULTRIX-11 Software Technical Description

NAME

ra - UDA50/RA60, RA80, RA81 winchester disks
rd/rx - RQDX1/RD51/RD52 winchester disk and RX50 floppy disk
rc - KLESI/RC25 winchester disk

DESCRIPTION

The RA disk driver supports the UDA50 disk controller , used with unibus PDP11 processors, and the RQDX1 Q bus disk controller. Both of these controllers support the Digital Storage Architecture (DSA) disks via the Mass Storage Control Protocol (MSCP).

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for information about logical disk names and Appendix D for disk partition sizes.

The ra files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records.

A 'raw' interface provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and 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 the buffer must begin on a word boundary.

FILES

/dev/ra##, /dev/rra##
/dev/rd#, /dev/rrd#
/dev/rx#, /dev/rrx#
/dev/rc#, /dev/rrc#

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

rk - RK11/RK03 or RK05 disk

DESCRIPTION

Rk# refers to an entire disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 4871. Minor device numbers are drive numbers on one controller.

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for more information about logical disk names and Appendix D for disk sizes.

The rk files discussed above access the disk via 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 results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RK files begin with rrk and end with a number which selects the same disk as the corresponding rk file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise seek calls should specify a multiple of 512 bytes.

FILES

/dev/rk#, /dev/rrk#

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

rl - RL11/RL01 or RL02 disk

DESCRIPTION

The RL11 controller supports a mix of up to four RL01/RL02 disks.

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for more information about logical disk names and Appendix D for disk sizes.

Rl# refers to an entire disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 10239 (RL01) or 0 to 20479 (RL02). The physical disk sector size is 128 words, however the logical block size is 256 words. Minor device numbers are drive numbers on one controller.

The rl files discussed above access the disk via 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 results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RL files begin with rrl and end with a number which selects the same disk as the corresponding rl file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise seek calls should specify a multiple of 512 bytes.

FILES

/dev/rl#, /dev/rrl#

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

rp - RP11/RP02 or RP03 moving-head disk

DESCRIPTION

The RP11 controller supports a mix of up to eight RP02/RP03 disk drives.

The files rp0 ... rp7 refer to sections of RP disk drive 0. The files rp8 ... rp15 refer to drive 1 etc. This allows a large disk to be broken up into more manageable pieces.

Refer to the ULTRIX-11 System Management Guide, Section 1.4 for more information about logical disk names and Appendix D for disk partition sizes.

The rp files access the disk via 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 results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with rrp and end with a number which selects the same disk section as the corresponding rp file.

In raw I/O the buffer must begin on a word boundary.

FILES

/dev/rp##, /dev/rrp##

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A

ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O read and write(2) truncate file offsets to 512-byte block boundaries, and write scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, read, write and lseek(2) should always deal in 512-byte multiples.

NAME

tc - TC11/TU56 DECtape

DESCRIPTION

The files tap0 ... tap7 refer to the TC-11/TU56 DECtape drives 0 to 7.

The 256-word blocks on a standard DECtape are numbered 0 to 577.

The unibus interrupt request priority level for the TC11 must be set to BR 6.

FILES

/dev/tap?

SEE ALSO

tp(1)

RESTRICTIONS

DECtape is an obsolete device and is not supported by ULTRIX-11.

NAME

tm - TM11 - TU10/TE10/TS03 800 BPI magtape interface

DESCRIPTION

The files mt0, ..., mt7 refer to the DEC TM11 magtape. When closed it can be rewound or not, see below. If it was open for writing, two end-of-files are written. If the tape is not to be rewound it is positioned with the head between the two tapemarks. If the 0200 bit is on in the minor device number the tape is not rewound when closed. Refer to the ULTRIX-11 System Management Guide, Section 1.4 for more information about magtape logical device names.

A standard tape consists of a series of 512 byte records terminated by an end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

The mt files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated files are named rmt0, ..., rmt7. Each read or write 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; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. 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.

FILES

/dev/mt#, /dev/rmt#, /dev/nrmt#

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A

ULTRIX-11 Software Technical Description

RESTRICTIONS

If any non-data error is encountered, it refuses to do anything more until closed. In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark.

NAME

ts - TS11/TSV05/TU80/TK25 1600 BPI magtape interface

DESCRIPTION

The TS driver supports a single TS11 for unibus processors and a single TSV05 for Q bus processors.

The file ht# refers to the DEC TS11 magtape. When opened for reading or writing, the tape is not rewound. When closed, it is rewound (unless the 0200 bit is on, see below). If the tape was open for writing, a double end-of-file is written. If the tape is not to be rewound the tape is backspaced to just between the two tapemarks.

A standard tape consists of a series of 512 byte records terminated by a double end-of-file. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time. Writing in very small units is inadvisable, however, because it tends to create monstrous record gaps.

If the 0200 bit is on (initial digit 2 or 3), the tape is not rewound on close. Note that the minor device number has no necessary connection with the file name.

The ht files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the 'raw' interface is appropriate. The associated file may be named rht# but the same minor-device considerations as for the regular files still apply.

Each read or write 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; if the record is long, an error is indicated. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. A zero count is returned when a tape mark is read; another read will fetch the first record of the next tape file.

FILES

/dev/ht#, /dev/rht#, /dev/nrht#

SEE ALSO

ULTRIX-11 System Management Guide, Section 2.6 and Appendix A
ULTRIX-11 Software Technical Description

RESTRICTIONS

In raw I/O, there should be a way to perform forward and backward record and file spacing and to write an EOF mark explicitly.

NAME

tty - general terminal interface

DESCRIPTION

This section describes both a particular special file /dev/tty and the terminal drivers used for conversational computing.

Line disciplines.

The system provides different line disciplines for controlling communications lines. In this version of the system there are two disciplines available:

old — The old (standard) terminal driver. This is used when using the standard shell sh(1) and for compatibility with other standard version 7 UNIX systems.

new — A newer terminal driver, with features for job control; this must be used when using csh(1). See newtty(1) for a short user-level summary.

Line discipline switching is accomplished with the TIOCSETD ioctl:

```
int ldisc = LDISC; ioctl(filedes, TIOCSETD, &ldisc);
```

where LDISC is OTTYDISC for the standard tty driver and NTTYDISC for the new driver. The standard (currently old) tty driver is discipline 0 by convention. The current line discipline can be obtained with the TIOCGETD ioctl. Pending input is discarded when the line discipline is changed.

All of the low-speed asynchronous communications ports can use any of the available line disciplines, no matter what hardware is involved.

The control terminal.

When a terminal file is opened, it causes the process to wait until a connection is established. In practice, user programs seldom open these files; they are opened by init(8) and become a user's standard input and output file.

If a process which has no control terminal opens a terminal file, then that terminal file becomes the control terminal for that process. The control terminal is thereafter inherited by a child process during a fork(2), even if the control terminal is closed.

The file /dev/tty is, in each process, a synonym for a control terminal associated with that process. It is useful

for programs 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 a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

Process groups.

As described more completely in jobs(3), command processors such as csh(1) can arbitrate the terminal between different jobs by placing related jobs in a single process group and associating this process group with the terminal. A terminal's associated process group may be set using the TIOCSPGRP ioctl(2):

```
ioctl(fildes, TIOCSPGRP, &pgrp)
```

or examined using TIOCGPGRP rather than TIOCSPGRP, returning the current process group in pgrp. The new terminal driver aids in this arbitration by restricting access to the terminal by processes which are not in the current process group; see Job access control below.

Modes.

The terminal drivers have three major modes, characterized by the amount of processing on the input and output characters:

cooked — The normal mode. In this mode lines of input are collected and input editing is done. The edited line is made available when it is completed by a newline or when an EOT (control-D, hereafter ^D) is entered. A carriage return is usually made synonymous with newline in this mode, and replaced with a newline whenever it is typed. All driver functions (input editing, interrupt generation, output processing such as delay generation and tab expansion, etc.) are available in this mode.

CBREAK — This mode eliminates the character, word, and line editing input facilities, making the input character available to the user program as it is typed. Flow control, literal-next and interrupt processing are still done in this mode. Output processing is done.

RAW — This mode eliminates all input processing and makes all input characters available as they are typed; no output processing is done either. TANDEM mode is available for input flow control, however; see below. Characters are a full 8 bits

TTY(4)

(parity is up to the program).

The style of input processing can also be very different when, in the new terminal driver, a process asks for notification via a SIGTTIN signal(2) when input is ready to be read from the control terminal. In this case a read(2) from the control terminal will never block, but rather return an error indication (EIO) if there is no input available.

Input editing.

A UNIX terminal 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 choked, 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. In the old terminal driver all the saved characters are thrown away when the limit is reached, without notice; the new driver simply refuses to accept any further input, and rings the terminal bell.

Input characters are normally accepted in either even or odd parity with the parity bit being stripped off before the character is given to the program. By clearing either the EVEN or ODD bit in the flags word it is possible to have input characters with that parity discarded (see the Summary below.)

In all of the line disciplines, it is possible to simulate terminal input using the TIOCSTI ioctl, 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 control terminal except when executed by the superuser (this call is not in standard version 7 UNIX).

Input characters are normally echoed by putting them in an output queue as they arrive. This may be disabled by clearing the ECHO bit in the flags word using the stty(2) call or the TIOCSETN or TIOCSETP ioctls (see the Summary below).

In cooked mode, terminal input is processed in units of lines. A program attempting to read will normally be suspended until an entire line has been received (but see the description of SIGTTIN in Modes above and FIONREAD in Summary below.) 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.

During input, line editing is normally done, with the character '#' logically erasing the last character typed and the character '@' logically erasing the entire current input line. These are often reset on crt's, with ^H replacing #, and ^U replacing @. These characters never erase beyond the beginning of the current input line or an ^D. These characters may be entered literally by preceding them with '\'; in the old teletype driver both the '\' and the character entered literally will appear on the screen; in the new driver the '\' will normally disappear.

The drivers normally treat either a carriage return or a newline character as terminating an input line, replacing the return with a newline and echoing a return and a line feed. If the CRMOD bit is cleared in the local mode word then the processing for carriage return is disabled, and it is simply echoed as a return, and does not terminate cooked mode input.

In the new driver there is a literal-next character ^V which can be typed in both cooked and CBREAK mode preceding any character to prevent its special meaning. This is to be preferred to the use of '\' escaping erase and kill characters, but '\' is (at least temporarily) retained with its old function in the new driver for historical reasons.

The new terminal driver also provides two other editing characters in normal mode. The word-erase character, normally ^W, erases the preceding word, but not any spaces before it. For the purposes of ^W, a word is defined as a sequence of non-blank characters, with tabs counted as blanks. Finally, the reprint character, normally ^R, retypes the pending input beginning on a new line. Retyping occurs automatically in cooked mode if characters which would normally be erased from the screen are fouled by program output.

Input echoing and redisplay

In the old terminal driver, nothing special occurs when an erase character is typed; the erase character is simply echoed. When a kill character is typed it is echoed followed by a new-line (even if the character is not killing the line, because it was preceded by a '\'!).

The new terminal driver has several modes for handling the echoing of terminal input, controlled by bits in a local mode word.

Hardcopy terminals. When a hardcopy terminal is in use, the LPRTERA bit is normally set in the local mode word. Characters which are logically erased are then printed out

TTY(4)

backwards preceded by '\' and followed by '/' in this mode.

Crt terminals. When a crt terminal is in use, the LCRTBS bit is normally set in the local mode word. The terminal driver then echoes the proper number of erase characters when input is erased; in the normal case where the erase character is a ^H this causes the cursor of the terminal to back up to where it was before the logically erased character was typed. If the input has become fouled due to interspersed asynchronous output, the input is automatically retyped.

Erasing characters from a crt. When a crt terminal is in use, the LCRTERA bit may be set to cause input to be erased from the screen with a "backspace-space-backspace" sequence when character or word deleting sequences are used. A LCRTKIL bit may be set as well, causing the input to be erased in this manner on line kill sequences as well.

Echoing of control characters. If the LCTLECH bit is set in the local state word, then non-printing (control) characters are normally echoed as ^X (for some X) rather than being echoed unmodified; delete is echoed as ^?.

The normal modes for using the new terminal driver on crt terminals are speed dependent. At speeds less than 1200 baud, the LCRTERA and LCRTKILL processing is painfully slow, so stty(1) normally just sets LCRTBS and LCTLECH; at speeds of 1200 baud or greater all of these bits are normally set. Stty(1) summarizes these option settings and the use of the new terminal driver as "newcrt."

Output processing.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. (As noted above, input characters are normally echoed by putting them in the output queue as they arrive.) When 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. Even parity is normally generated on output. The EOT character is not transmitted in cooked mode to prevent terminals that respond to it from hanging up; programs using raw or cbreak mode should be careful.

The terminal drivers provide necessary processing for cooked and CBREAK mode output including delay generation for certain special characters and parity generation. Delays are available after backspaces ^H, form feeds ^L, carriage returns ^M, tabs ^I and newlines ^J. The driver will also optionally expand tabs into spaces, where the tab stops are

assumed to be set every eight columns. These functions are controlled by bits in the tty flags word; see Summary below.

The terminal drivers provide for mapping between upper and lower case on terminals lacking lower case, and for other special processing on deficient terminals.

Finally, in the new terminal driver, there is a output flush character, normally ^O, which sets the LFLUSHO bit in the local mode word, causing subsequent output to be flushed until it is cleared by a program or more input is typed. This character has effect in both cooked and CBREAK modes and causes pending input to be retyped if there is any pending input. Ioctls to flush the characters in the input and output queues, TIOCFLUSH, and to return the number of characters still in the output queue, TIOCOUTQ are also available.

Upper case terminals and Hazeltines

If the LCASE bit is set in the tty flags, then all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by '\'. In addition, the following escape sequences can be generated on output and accepted on input:

for	`	|	~	{	}
use	\'	\!	\^	\(	\)

To deal with Hazeltine terminals, which do not understand that ~ has been made into an ASCII character, the LTILDE bit may be set in the local mode word when using the new terminal driver; in this case the character ~ will be replaced with the character ` on output.

Flow control.

There are two characters (the stop character, normally ^S, and the start character, normally ^Q) which cause output to be suspended and resumed respectively. Extra stop characters typed when output is already stopped have no effect, unless the start and stop characters are made the same, in which case output resumes.

A bit in the flags word may be set to put the terminal into TANDEM mode. In this mode the system produces a stop character (default ^S) when the input queue is in danger of overflowing, and a start character (default ^Q) when the input has drained sufficiently. This mode is useful when the terminal is actually another machine that obeys the conventions.

TTY(4)

Line control and breaks.

There are several ioctl calls available to control the state of the terminal line. The TIOCSBRK ioctl will set the break bit in the hardware interface causing a break condition to exist; this can be cleared (usually after a delay with sleep(3)) by TIOCCBRK. Break conditions in the input are reflected as a null character in RAW mode or as the interrupt character in cooked or CBREAK mode. The TIOCCDTR ioctl will clear the data terminal ready condition; it can be set again by TIOCSDTR. TIOCSBRK, TIOCBRK, TIOCSDTR and TIOCCDTR are available only where the hardware and device driver are able to support them.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a SIGHUP hangup signal is sent to the processes in the distinguished process group of the terminal; this usually causes them to terminate (the SIGHUP can be suppressed by setting the LNOHANG bit in the local state word of the driver.) Access to the terminal by other processes is then normally revoked, so any further reads will fail, and programs that read a terminal and test for end-of-file on their input will terminate appropriately.

It is possible to ask that the phone line be hung up on the last close with the TIOCHPCL ioctl; this is normally done on the outgoing line.

Interrupt characters.

There are several characters that generate interrupts in cooked and CBREAK mode; all are sent the processes in the control group of the terminal, as if a TIOCGPGRP ioctl were done to get the process group and then a killpg(2) system call were done, except that these characters also flush pending input and output when typed at a terminal (a 'la TIOCFLUSH). The characters shown here are the defaults; the field names in the structures (given below) are also shown. The characters may be changed, although this is not often done.

^? t_intrc (Delete) generates a SIGINTR signal. This is the normal way to stop a process which is no longer interesting, or to regain control in an interactive program.

^\ t_quitc (FS) generates a SIGQUIT signal. This is used to cause a program to terminate and produce a core image, if possible, in the file core in the current directory.

^Z t_suspc (EM) generates a SIGTSTP signal, which is used

to suspend the current process group.

^Y t_dstopc (SUB) generates a SIGTSTP signal as ^Z does, but the signal is sent when a program attempts to read the ^Y, rather than when it is typed.

Job access control.

When using the new terminal driver, if a process which is not in the distinguished process group of its control terminal attempts to read from that terminal its process group is sent a SIGTTIN signal, which normally causes the members of that process group to stop. If, however, the process is ignoring or holding SIGTTIN signal, is an orphan its parent has exited and it has been inherited by the init(8) process, or if it is a process in the middle of process creation using vfork(2)), it is instead returned an end-of-file. Under older UNIX systems these processes would typically have had their input files reset to /dev/null, so this is a compatible change.

When using the new terminal driver with the LTOSTOP bit set in the local modes, a process is prohibited from writing on its control terminal if it is not in the distinguished process group for that terminal. Processes which are holding or ignoring SIGTTOU signals, which are orphans, or which are in the middle of a vfork(2) are excepted and allowed to produce output.

Summary of modes.

Unfortunately, due to the evolution of the terminal driver, there are 4 different structures which contain various portions of the driver data. The first of these (sgttyb) contains that part of the information largely common between version 6 and version 7 UNIX systems. The second contains additional control characters added in version 7. The third is a word of local state peculiar to the new terminal driver, and the fourth is another structure of special characters added for the new driver. In the future a single structure may be made available to programs which need to access all this information; most programs need not concern themselves with all this state.

The basic ioctls use the structure defined in <sgtty.h>:

```
struct sgttyb {
     char sg_ispeed;
     char sg_ospeed;
     char sg_erase;
     char sg_kill;
     shortsg_flags;
};
```

};

The sg ispeed and sg ospeed fields describe the input and output speeds of the device according to the following table, which corresponds to the DEC DH-11 interface. If other hardware is used, impossible speed changes are ignored. Symbolic values in the table are as defined in <sgtty.h>.

B0	0	(hang up dataphone)
B50	1	50 baud
B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A
EXTB	15	External B

In the current configuration, only 110, 150, 300 and 1200 baud are really supported on dial-up lines. Code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program. The half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied; full-duplex 212 datasets work fine.

The sg erase and sg kill fields of the argument structure specify the erase and kill characters respectively. (Defaults are # and @.)

The sg flags field of the argument structure contains several bits that determine the system's treatment of the terminal:

ALLDELAY	0177400	Delay algorithm selection
BSDELAY	0100000	Select backspace delays (not implemented):
BS0	0	
BS1	0100000	
VTDELAY	0040000	Select form-feed and vertical-tab delays:
FF0	0	
FF1	0100000	
CRDELAY	0030000	Select carriage-return delays:
CR0	0	
CR1	0010000	
CR2	0020000	

CR3	0030000	
TBDELAY	0006000	Select tab delays:
TAB0	0	
TAB1	0001000	
TAB2	0004000	
XTABS	0006000	
NLDELAY	0001400	Select new-line delays:
NL0	0	
NL1	0000400	
NL2	0001000	
NL3	0001400	
EVENP	0000200	Even parity allowed on input (most terminals)
ODDP	0000100	Odd parity allowed on input
RAW	0000040	Raw mode: wake up on all characters, 8-bit interface
CRMOD	0000020	Map CR into LF; echo LF or CR as CR-LF
ECHO	0000010	Echo (full duplex)
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Return each character as soon as typed
TANDEM	0000001	Automatic flow control

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.

Backspace delays are currently ignored but might be used for Terminet 300's.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is suitable for the concept-100 and pads lines to be at least 9 characters at 9600 baud.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

Input characters with the wrong parity, as determined by bits 0200 and 0100, are ignored in cooked and CBREAK mode.

TTY(4)

RAW disables all processing save output flushing with LFLUSHO; full 8 bits of input are given as soon as it is available; all 8 bits are passed on output. A break condition in the input is reported as a null character. If the input queue overflows in raw mode it is discarded; this applies to both new and old drivers.

CRMOD causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (for terminals with a new-line function).

CBREAK is a sort of half-cooked (rare?) mode. Programs can read each character as soon as typed, instead of waiting for a full line; all processing is done except the input editing: character and word erase and line kill, input reprint, and the special treatment of \ or EOT are disabled.

TANDEM mode causes the system to produce a stop character (default ^S) whenever the input queue is in danger of overflowing, and a start character (default ^Q) when the input queue has drained sufficiently. It is useful for flow control when the 'terminal' is really another computer which understands the conventions.

In addition to the TIOCSETD and TIOCGETD disciplines discussed in Line disciplines above, a large number of other ioctl(2) calls apply to terminals, and have the general form:

```
#include <sgtty.h>

ioctl(fildes, code, arg)
struct sgttyb *arg;
```

The applicable codes are:

TIOCGETP — Fetch the basic parameters associated with the terminal, and store in the pointed-to sgttyb structure.

TIOCSETP — Set the parameters according to the pointed-to sgttyb structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOCSETN — Set the parameters like TIOCSETP but do not delay or flush input. Input is not preserved, however, when changing to or from RAW.

TTY(4)

With the following codes the arg is ignored.

TIOCEXCL — Set "exclusive-use" mode: no further opens are permitted until the file has been closed.

TIOCNXCL — Turn off "exclusive-use" mode.

TIOCHPCL — When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with an ACU used to place outgoing calls.

The remaining calls are not available in vanilla version 7 UNIX. In cases where arguments are required, they are described; arg should otherwise be given as 0.

TIOCFLUSH — If arg is 0, all characters waiting in input or output queues are flushed. If arg is FREAD (defined in <sys/file.h>), all characters in the input queues are flushed. If arg is FWRITE (defined in <sys/file.h>), all characters in the output queues are flushed.

TIOCSTI — The argument is the address of a character which the system pretends was typed on the terminal.

TIOCSBRK — The break bit is set in the terminal.

TIOCCBRK — The break bit is cleared.

TIOCSDTR — Data terminal ready is set.

TIOCCDTR — Data terminal ready is cleared.

TIOCGPGRP — Arg is the address of a word into which is placed the process group number of the control terminal.

TIOCSPGRP — Arg is a word (typically a process id) which becomes the process group for the control terminal.

FIONREAD — Returns in the long integer whose address is arg the number of immediately readable characters from the argument unit. This works for files, pipes, and terminals, but not (yet) for multiplexed channels.

The second structure associated with each terminal specifies characters that are special in both the old and new terminal interfaces: The following structure is defined in

<sys/ioctl.h>, which is automatically included in <sgtty.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 default values for these characters are ^?, ^\, ^Q, ^S, ^D, and -1. A character value of -1 eliminates the effect of that character. The t_brkc character, by default -1, acts like a new-line in that it terminates a 'line,' is echoed, and is passed to the program. The 'stop' and 'start' characters may be the same, to produce a toggle effect. It is probably counterproductive to make other special characters (including erase and kill) identical. The applicable ioctl calls are:

TIOCGETC Get the special characters and put them in the specified structure.

TIOCSETC Set the special characters to those given in the structure.

The third structure associated with each terminal is a local mode word; except for the LNOHANG bit, this word is interpreted only when the new driver is in use. The bits of the local mode word are:

LCRTBS	000001	Backspace on erase rather than echoing erase
LPRTERA	000002	Printing terminal erase mode
LCRTERA	000004	Erase character echos as backspace-space-backspace
LTILDE	000010	Convert ~ to on output (for Hazeltine terminals)
LMDMBUF	000020	Stop/start output when carrier drops
LLITOUT	000040	Suppress output translations
LTOSTOP	000100	Send SIGTTOU for background output
LFLUSHO	000200	Output is being flushed
LNOHANG	000400	Don't send hangup when carrier drops
LETXACK	001000	Diablo style buffer hacking (unimplemented)
LCRTKIL	002000	BS-space-BS erase entire line on line kill
LINTRUP	004000	Generate interrupt SIGTINT when input ready to read
LCTLECH	010000	Echo input control chars as ^X, delete as ^?
LPENDIN	020000	Retype pending input at next read or input character
LDECCTQ	040000	Only ^Q restarts output after ^S, like DEC

systems

The applicable ioctl functions are:

TIOCLBIS — Arg is the address of a mask which is the bits to be set in the local mode word.

TIOCLBIC — Arg is the address of a mask of bits to be cleared in the local mode word.

TIOCLSET — Arg is the address of a mask to be placed in the local mode word.

TIOCLGET — Arg is the address of a word into which the current mask is placed.

The final structure associated with each terminal is the ltchars structure which defines interrupt characters for the new terminal driver. Its structure is:

```
struct ltchars {
     char t_suspc;  /* stop process signal */
     char t_dstopc; /* delayed stop process signal */
     char t_rprntc; /* reprint line */
     char t_flushc; /* flush output (toggles) */
     char t_werasec;/* word erase */
     char t_lnextc; /* literal next character */
};
```

The default values for these characters are ^Z, ^Y, ^R, ^O, ^W, and ^V. A value of -1 disables the character.

The applicable ioctl functions are:

TIOCSLTC — Arg is the address of an ltchars structure which defines the new local special characters.

TIOCGLTC — Arg is the address of an ltchars structure into which is placed the current set of local special characters.

FILES

/dev/tty
/dev/tty??
/dev/console

SEE ALSO

csh(1), stty(1), ioctl(2), signal(2), sigsys(2j), stty(2), newtty(4), getty(8), init(8)

RESTRICTIONS

Half-duplex terminals are not supported.

NAME

a.out - assembler and link editor output

SYNOPSIS

```
#include <a.out.h>
```

DESCRIPTION

A.out is the output file of the assembler as(1) and the link editor ld(1). Both programs make a.out executable if there were no errors and no unresolved external references. Layout information as given in the include file for the PDP11 is:

```
/*
 * SCCSID: %W%   %G%
 */
struct  exec {   /* a.out header */
        int      a_magic;   /* magic number */
        unsigned a_text;    /* size of text segment */
        unsigned a_data;    /* size of initialized data */
        unsigned a_bss;     /* size of unitialized data */
        unsigned a_syms;    /* size of symbol table */
        unsigned a_entry;   /* entry point */
        unsigned a_unused;  /* not used */
        unsigned a_flag;    /* relocation info stripped */
};

#define NOVL     7

struct  ovlhdr {
        int                 max_ovl;/* maximum ovl size */
        unsigned ov_siz[NOVL];/* size of i'th overlay */
};

#define A_MAGIC1 0407        /* normal (I space only) */
#define A_MAGIC2 0410        /* read-only/shared text */
#define A_MAGIC3 0411        /* separated I&D */
#define A_MAGIC4 0405        /* overlay (NOT USED) */
#define A_MAGIC5 0430        /* overlay text kernel */
#define A_MAGIC6 0430        /* user overlay (shared text) */
#define A_MAGIC7 0431        /* user overlay (separated I&D) */

struct  nlist {  /* symbol table entry */
        char     n_name[8]; /* symbol name */
        int      n_type;    /* type flag */
        unsigned n_value;   /* value */
};
/*
 * Macros that take exec structures as arguments and tell
 * whether the file has a reasonable magic number or
 * offset to text.
 */
```

A.OUT(5)

```
#define N_BADMAG(x) \
  (((x).a_magic)!=A_MAGIC1 && ((x).a_magic)!=A_MAGIC2 && \
   ((x).a_magic)!=A_MAGIC3 && ((x).a_magic)!=A_MAGIC4 && \
   ((x).a_magic)!=A_MAGIC5 && ((x).a_magic)!=A_MAGIC6 && \
   ((x).a_magic)!=A_MAGIC7)

#define N_OVMAG(x) \
  (((x).a_magic)==A_MAGIC5 || ((x).a_magic)==A_MAGIC6 || \
   ((x).a_magic)==A_MAGIC7)

#define N_TXTOFF(x) \
   (N_OVMAG(x) ? sizeof (struct ovlhdr) + sizeof (struct exec) \
       : sizeof (struct exec))

                /* values for type flag */
#define N_UNDF   0          /* undefined */
#define N_ABS    01         /* absolute */
#define N_TEXT   02         /* text symbol */
#define N_DATA   03         /* data symbol */
#define N_BSS    04         /* bss symbol */
#define N_TYPE   037
#define N_REG    024        /* register name */
#define N_FN     037        /* file name symbol */
#define N_EXT    040        /* external bit, or'ed in */
#define FORMAT   "%06o"     /* to print a value */
```

The file has four sections: a header, the program and data text, relocation information, and a symbol table (in that order). The last two may be empty if the program was loaded with the '-s' option of ld or if the symbols and relocation have been removed by strip(1).

In the header the sizes of each section are given in bytes, but are even. The size of the header is not included in any of the other sizes.

When an a.out file is loaded into core for execution, 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. The text segment begins at 0 in the core image; the header is not loaded. If the magic number in the header is 0407(8), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. If the magic number is 0410, the data segment begins at the first 0 mod 8K byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. If the magic number is 411, the text segment is again pure, write-protected, and shared, and moreover instruction and data space are

A.OUT(5)

separated; the text and data segment both begin at location 0. If the magic number is 0405, the text segment is overlaid on an existing (0411 or 0405) text segment and the existing data segment is preserved.

The stack will occupy the highest possible locations in the core image: from 0177776(8) and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by brk(2).

The start of the text segment in the file is 020(8); the start of the data segment is 020+St (the size of the text) the start of the relocation information is 020+St+Sd; the start of the symbol table is 020+2(St+Sd) if the relocation information is present, 020+St+Sd if not.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file. Other flag values may occur if an assembly language program defines machine instructions.

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.

The value of a word in the text or data portions which is not a reference to an undefined external symbol is exactly that value which will appear in core when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information for that word, then the value of the word as 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 will be added into the word in the file.

If relocation information is present, it amounts to one word per word of program text or initialized data. There is no relocation information if the 'relocation info stripped' flag in the header is on.

Bits 3-1 of a relocation word indicate the segment referred to by the text or data word associated with the relocation word:

000 absolute number
002 reference to text segment
004 reference to initialized data
006 reference to uninitialized data (bss)
010 reference to undefined external symbol

A.OUT(5)

Bit 0 of the relocation word indicates, if 1, that the reference is relative to the pc (e.g., 'clr x'); if 0, that the reference is to the actual symbol (e.g., 'clr *$x').

The remainder of the relocation word (bits 15-4) contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

SEE ALSO

as(1), ld(1), nm(1)

ACCT(5)

NAME

acct - execution accounting file

SYNOPSIS

```
#include <sys/acct.h>
```

DESCRIPTION

Acct(2) causes entries to be made into 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:

```
/*
 * SCCSID: @(#)acct.h        1.18/2/84
 */
/*
 * Accounting structures
 */

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 */
        short   ac_uid;      /* Accounting user ID */
        short   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 */
};

extern  struct  acct acctbuf;
extern  struct  inode *acctp;/* inode of accounting file */

#define AFORK   01           /* has executed fork, but no exec */
#define ASU     02           /* used super-user privileges */
```

If the process does an exec(2), the first 10 characters of the filename appear in ac comm. The accounting flag contains bits indicating whether exec(2) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

acct(2), sa(1)

AR(5)

NAME

ar - archive (library) file format

SYNOPSIS

```
#include <ar.h>
```

DESCRIPTION

The archive command ar is used to combine several files into one. Archives are used mainly as libraries to be searched by the link-editor ld.

A file produced by ar has a magic number 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:

```
/*
 * SCCSID: @(#)ar.h1.011/12/83
 */
#define ARMAG 0177545
struct  ar_hdr {
        char  ar_name[14];
        long  ar_date;
        char  ar_uid;
        char  ar_gid;
        int   ar_mode;
        long  ar_size;
};
```

The name is a null-terminated string; the date is in the form of time(2); the user ID and group ID are numbers; the mode is a bit pattern per chmod(2); the size is counted in bytes.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

SEE ALSO

ar(1), ld(1), nm(1)

RESTRICTIONS

Coding user and group IDs as characters is a botch.

CORE(5)

NAME

core - format of core image file

DESCRIPTION

UNIX writes out a core image of a terminated process when any of various errors occur. See signal(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called 'core' and is written in the process's working directory (provided it can be; normal access controls apply).

The first 1024 bytes of the core image are a copy of the system's per-user data for the process, including the registers as they were at the time of the fault; see the system listings for the format of this area. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is write-protected and shared, it is not dumped; otherwise the entire address space is dumped.

In general the debugger adb(1) is sufficient to deal with core images.

SEE ALSO

adb(1), signal(2)

CPIO(5)

NAME

cpio - format of cpio archive

DESCRIPTION

The header structure, when the -c option of cpio(1) is not used, is:

```
struct {
        short   h_magic,
                h_dev;
        u_short 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 word];
} Hdr;
```

When the -c option is used, the header information is described by:

```
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,
        &Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);
```

Longtime and Longfile are equivalent to Hdr.h_mtime and Hdr.h_filesize, respectively. The contents of each file are 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.

SEE ALSO

cpio(1), find(1), stat(2).

NAME

dir - format of directories

SYNOPSIS

```
#include <sys/dir.h>
```

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry see, filsys(5). The structure of a directory entry as given in the include file is:

```
/*
 * SCCSID: @(#)dir.h1.011/12/83
 */
#ifndef DIRSIZ
#define DIRSIZ14
#endif
struct  direct
{
        ino_t d_ino;
        char  d_name[DIRSIZ];
};
```

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 and for the root directories of removable file systems. In the first case, there is no parent, and in the second, the system does not permit off-device references. Therefore in both cases '..' has the same meaning as '.'.

SEE ALSO

filsys(5)

DUMP(5)

NAME

dump, ddate - incremental dump format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ino.h>
#include <dumprestor.h>
```

DESCRIPTION

Tapes used by dump and restor(1) 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:

```
/*
 * SCCSID: @(#)dumprestor.h1.011/12/83
 */
#define NTREC     20
#define MLEN      16
#ifdef            RESTOR40
#define           MSIZ3072
#else
#define MSIZ      4096
#endif

#define TS_TAPE   1
#define TS_INODE  2
#define TS_BITS   3
#define TS_ADDR   4
#define TS_END    5
#define TS_CLRI   6
#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       dinodec_dinode;
     int          c_count;
     char         c_addr[BSIZE];
```

```
} spcl;

struct          idates
{
     char       id_name[16];
     char       id_incno;
     time_t     id_ddate;
};
```

NTREC is the number of 512 byte records in a physical tape block. 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 (512-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 filsys(5).

c_count The count of characters in c_addr.

DUMP(5)

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 of the file /etc/ddate 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(1).
id_ddate The date of the incremental dump in system format see types(5).

FILES

/etc/ddate

SEE ALSO

dump(1), dumpdir(1), restor(1), filsys(5), types(5)

NAME

environ - user environment

SYNOPSIS

```
extern char **environ;
```

DESCRIPTION

An array of strings called the 'environment' is made available by exec(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, time, nice(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by ':'. Login(1) sets PATH=:/bin:/usr/bin.

HOME A user's login directory, set by login(1) from the password file passwd(5).

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as nroff or plot(1), which may exploit special terminal capabilities. See term(7) for a list of terminal types.

Further names may be placed in the environment by the export command and 'name=value' arguments in sh(1), or by exec(2). It is unwise to conflict with certain Shell variables that are frequently exported by '.profile' files: MAIL, PS1, PS2, IFS.

SEE ALSO

exec(2), sh(1), term(7), login(1), termcap(5)

NAME

filsys, flblk, ino - format of file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/flbk.h>
#include <sys/filsys.h>
#include <sys/ino.h>
```

DESCRIPTION

Every file system storage volume (e.g. RL disk, RP disk, HP disk, DECtape reel) has a common format for certain vital information. Every such volume is divided into a certain number of 512-byte blocks. Block 0 is unused and is available to contain a bootstrap program, pack label, or other information.

Block 1 is the super block. The layout of the super block as defined by the include file <sys/filsys.h> is:

```
/*
 * SCCSID: @(#)filsys.h  1.3  8/18/84
 */
#include  <sys/localopts.h>
/*
 * Structure of the super-block
 */
struct    filsys {
     unsigned short s_isize; /* size in blocks of i-list */
     daddr_t   s_fsize;       /* size in blocks of entire volume */
     short     s_nfree;       /* number of addresses in s_free */
     daddr_t   s_free[NICFREE];/* free block list */
     short     s_ninode;      /* number of i-nodes in s_inode */
     ino_t     s_inode[NICINOD];/* free i-node list */
     char      s_flock;       /* lock during free list
                                 manipulation */
     char      s_ilock;       /* lock during i-list manipulation */
     char      s_fmod;        /* super block modified flag */
     char      s_ronly;       /* mounted read-only flag */
     time_t    s_time;        /* last super block update */
     /* set by mkfs, updated by fsck, not used by system */
     daddr_t   s_tfree;       /* total free blocks*/
     ino_t     s_tinode;      /* total free inodes */
     /* set by mkfs, used by fsck to salvage free lists */
     short   s_m;   /* interleave - free list spacing */
     short     s_n; /* interleave - blocks per cylinder */
     /* set by mkfs and/or labelit, printed by fsck
           and labelit */
     char      s_fname[6];    /* file system name */
     char      s_fpack[6];    /* file system pack name */
```

```
#ifdef    BUFFERS
     ino_t     s_lasti;  /* start place for circular search */
     ino_t     s_nbehind;     /* est # free inodes before s_lasti */
#endif    BUFFERS
};

#ifdef    MAPMOUNT
/*
 * flags to avoid locking problems in alloc, free, ialloc
 */
#define   S_BUSY          01   /* alloc in progress */
#define   S_WANTED  02    /* want to alloc */
#endif
```

S_isize is the address of the first block after the i-list, which starts just after the super-block, in block 2. Thus is i-list is s_isize-2 blocks long. S_fsize is the address of the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block addresses; if an 'impossible' block address is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The s_free array contains, in s_free[1], ... , s_free[s_nfree-1], up to NICFREE free block numbers. NICFREE is a configuration constant. S_free[0] is the block address of the head of a chain of blocks constituting the free list. The layout of each block of the free chain as defined in the include file <sys/fblk.h> is:

```
/*
 * SCCSID: @(#)fblk.h    1.0  11/12/83
 */
struct fblk
{
     int       df_nfree;
     daddr_t   df_free[NICFREE];
};
```

The fields df_nfree and df_free in a free block are used exactly like s_nfree and s_free in the super block. To allocate a block: decrement s_nfree, and the new block number is s_free[s_nfree]. If the new block address is 0, there are no blocks left, so give an error. If s_nfree became 0, read the new block into s_nfree and s_free. To free a block, check if s_nfree is NICFREE; if so, copy s_nfree and the s_free array into it, write it out, and set s_nfree to 0. In any event set s_free[s_nfree] to the freed

block's address and increment s nfree.

S ninode is the number of free i-numbers in the s inode array. To allocate an i-node: if s ninode is greater than 0, decrement it and return s inode[s ninode]. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the s inode array, then try again. To free an i-node, provided s ninode is less than NICINODE, place its number into s inode[s ninode] and increment s ninode. If s ninode is already NICINODE, don't bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

S flock and s ilock are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of s fmod on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information. S ronly is a write-protection indicator; its disk value is also immaterial.

S time is the last time the super-block of the file system was changed. During a reboot, s time of the super-block for the root file system is used to set the system's idea of the time.

The fields s tfree, s tinode, s fname and s fpack are not currently maintained.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. I-nodes are 64 bytes long, so 8 of them fit into a block. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. The format of an i-node as given in the include file <sys/ino.h> is:

```
/*
 * SCCSID: @(#)ino.h    1.0     11/12/83
 */
/*
 * Inode structure as it appears on
 * a disk block.
 */
struct dinode
{
        unsigned short  di_mode;/* mode and type of file */
        short   di_nlink;       /* number of links to file */
        short   di_uid;         /* owner's user id */
        short   di_gid;         /* owner's group id */
```

```
        off_t   di_size;        /* number of bytes in file */
        char    di_addr[40];    /* disk block addresses */
        time_t  di_atime;       /* time last accessed */
        time_t  di_mtime;       /* time last modified */
        time_t  di_ctime;       /* time created */
};
#define INOPB   8       /* 8 inodes per block */
/*
 * the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

Di mode tells the kind of file; it is encoded identically to the st mode field of stat(2). Di nlink is the number of directory entries (links) that refer to this i-node. Di uid and di gid are the owner's user and group IDs. Size is the number of bytes in the file. Di atime and di mtime are the times of last access and modification of the file contents (read, write or create) (see times(2)); Di ctime records the time of last modification to the inode or to the file, and is used to determine whether it should be dumped.

Special files are recognized by their modes and not by i-number. A block-type special file is one which can potentially be mounted as a file system; a character-type special file cannot, though it is not necessarily character-oriented. For special files, the di addr field is occupied by the device code (see types(5)). The device codes of block and character special files overlap.

Disk addresses of plain files and directories are kept in the array di addr packed into 3 bytes each. The first 10 addresses specify device blocks directly. The last 3 addresses are singly, doubly, and triply indirect and point to blocks of 128 block pointers. Pointers in indirect blocks have the type daddr t (see types(5)).

For block b in a file to exist, it is not necessary that all blocks less than b exist. A zero block number either in the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated. Such a missing block reads as if it contained all zero words.

SEE ALSO

icheck(1), dcheck(1), dir(5), mount(1), stat(2), types(5)
ULTRIX-11 System Management Guide, Chapter 1

FSTAB(5)

NAME

fstab - file system table

DESCRIPTION

The fstab contains information about file systems. It is used by various system commands to obtain a default argument list and to control mounting and dismounting of file systems.

Each line of the fstab describes one file system, as follows:

special:directory:mode

Special is the special filename for the file system. Directory is the name of the directory where the file system will be mounted. Mode is the access mode of the file system, read/write or read only. Setting the mode to 'xx' causes the entry to be ignored. The order of entries in the fstab is important. Read ULTRIX-11 System Management Guide, Section 4.6.8 for a more complete description of the fstab and its usage.

FILES

/etc/fstab

SEE ALSO

df(1m), du(1), fsck(1m), mount(1m), quot(1m)
ULTRIX-11 System Management Guide, Section 4.1

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 neither an entry in the table for that name, nor one in the special default table.

Name	Type	Default	Description
ab	bool	false	auto-baud on carriage-return instead of break
ap	bool	false	terminal uses 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
co	bool	false	console - add \n after login prompt
ds	str	^Y	delayed suspend character
ec	bool	false	leave echo OFF
ep	bool	false	terminal uses even parity
er	str	^?	erase character
et	str	^D	end of text (EOF) character
fd	num	0	form-feed (vertical motion) delay
fl	str	^O	output flush character
hc	bool	false	do NOT hangup line on last close
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

GETTYTAB(5)

lc	bool	false	terminal has lower case
lm	str	login:	login prompt
ln	str	^V	"literal next" character
lo	str	/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 odd parity
os	num	unused	output speed
pe	bool	false	use printer (hard copy) erase algorithm
qu	str	^\	quit character
rp	str	^R	line retype character
rw	bool	false	do NOT use raw for input, use cbreak
sp	num	1200	line speed (input and output)
su	str	^Z	suspend character
tc	str	none	table continuation
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 will override line speed for stated direction only.

Should getty receive a null character (presumed to indicate a line break) it will rescan using the table name indicated by the nx entry. If there is none, it will re-use its original table name. Using ab, getty will autobaud on a carriage-return instead of an input break. This is useful for lines that are connected via a switch.

Delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

Output from getty is even parity unless op 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.

SEE ALSO

termcap(5), getty(8).

NAME

group - group file

DESCRIPTION

Group contains for each group the following information:

group name
encrypted password
numerical group ID
a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; Each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

newgrp(1), crypt(3), passwd(1), passwd(5)

MTAB(5)

NAME

mtab - mounted file system table

DESCRIPTION

Mtab resides in directory /etc and contains a table of devices mounted by the mount command. Umount removes entries.

Each entry is 64 bytes long; the first 32 are the null-padded name of the place where the special file is mounted; the second 32 are the null-padded name of the special file. The special file has all its directories stripped away; that is, everything through the last '/' is thrown away.

This table is present only so people can look at it. It does not matter to mount if there are duplicated entries nor to umount if a name cannot be found.

FILES

/etc/mtab

SEE ALSO

mount(1)

NAME

passwd - password file

DESCRIPTION

Passwd contains for each user the following information:

name (login name, contains no upper case)
encrypted password
numerical user ID
numerical group ID
GCOS job number, box number, optional GCOS user-id
initial working directory
program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The GCOS field is used only when communicating with that system, and in other installations can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

FILES

/etc/passwd

SEE ALSO

getpwent(3), login(1), crypt(3), passwd(1), group(5)

PHONES(5)

NAME

phones - remote host phone number data base

DESCRIPTION

The file /etc/phones contains the system-wide private phone numbers for the tip(1C) program. This file is normally unreadable, and so may contain privileged information. The format of the file 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.

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(3), and are interpreted for various devices by commands described in plot(1). 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(3).

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(1).

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.

PLOT(5)

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of plot(1). 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 isn't 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

plot(1), plot(3), graph(1)

NAME

printcap - printer capability data base

SYNOPSIS

/etc/printcap

DESCRIPTION

Printcap is a simplified version of the termcap(5) data base used to describe line 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 is used to describe 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 an option, -Pprinter, to allow explicit naming of a destination printer.

CAPABILITIES

Refer to termcap(5) for a description of the file layout.

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)
dn	str	'/usr/lib/lpd'	name of printer daemon
fc	num	0	if lp is a tty, clear flag bits (sgtty.h)
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
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
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nc	bool	false	suppress printing of control characters when using an output filter
of	str	NULL	name of output filtering program
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
rw	bool	false	open the printer device for reading and writing
sd	str	'/usr/spool/lpd'	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header
tr	str	NULL	trailer string to print when queue empties

PRINTCAP(5)

xc	num	0	if lp is a tty, clear local mode bits (tty (4))
xs	num	0	like 'xc' but set bits

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

SEE ALSO

lpq(1), lpr(1), lprm(1), termcap(5), lpd(8)

NAME

remote - remote host description file

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; e.g. 'dv=/dev/harris'. A numeric capability is specified by capability#value; e.g. '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.

ds (num) The baud rate used for dialing to the remote host, Used for the DF03 modem which can have the dial

REMOTE(5)

speed different than the baud rate. This is a decimal number. By default this is the same as the baud rate.

du (bool) This host is on a dial-up line.

dv (str) UNIX device(s) to open to establish a connection. 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. '~' escapes are only recognized by tip 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 end-of-file marks. The default is NULL.

oe (str) Output end-of-file string. The default is NULL. When tip is transferring a file, this string is sent at end-of-file.

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 even parity.

pn (str) Telephone number(s) for this host. If the telephone number field contains an @ sign, tip searches the file /etc/phones file for a list of telephone numbers; c.f. phones(5).

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/cau0: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

SEE ALSO
tip(1C), phones(5)

SCCSFILE(5)

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 user 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 user IDs of users who may add deltas to the file, separated by newlines. The lines containing these login names and/or numerical user 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(I) 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
```

The 't' flag defines the replacement for the %Y% 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 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 %M% 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

SCCSFILE(5)

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 (e.g., 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.

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 don't 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

get(1), delta(1), admin(1), prt(1)

An Introduction to the Source Code Control System by Eric Allman.

NAME

tar - tape archive format

DESCRIPTION

The tar command dumps files to and extracts files from magtape in tape archive format. Unlike tp tapes, tar tapes have no provision for a bootstrap program. The record size for tar tapes is always a multiple of 512 bytes. The most common blocking factors are one and 20.

Tape archive format consists of a header record followed by the contents of the file. The header record has the following structure:

```
#define TBLOCK 512
#define NBLOCK 20
#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];
          char rdev[6];  /* file type */
     } dbuf;
} dblock, tbuf[NBLOCK];
```

SEE ALSO

tar(1)

NAME

termcap - terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals, used, e.g., by vi(1) and curses(3). Terminals are described in termcap by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in termcap.

Entries in termcap consist of a number of ':' separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by '|' characters. The first name is always 2 characters long and is used by older version 6 systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

CAPABILITIES

(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

Name	Type	Pad?	Description
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		Backspace if not ^H
bs	bool		Terminal can backspace with ^H
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command char in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horiz motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed

dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give ":ei=:" if ic
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no cm)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing is
im	bool		Insert mode (enter); give ":im=:" if ic
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by "other" function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of "keypad transmit" mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of "other" keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in "keypad transmit" mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on "other" function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no cm)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Ok to move while in standout & underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correct working carriage rtn (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default \n)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with is)
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode

sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use cm
ti	str		String to begin programs that use cm
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Term. underlines even though doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like ce \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the termcap file as of this writing. (This particular concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\E1\ENH\EK\E\200\Eo&\200:\
 :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :\
 :co#80:\
 :dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:\
 :li#24:mi:nd=\E=:\
 :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and that empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in termcap are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has "automatic margins" (i.e. 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 co which indicates the number of columns the terminal has gives the value '80' for the Concept.

Finally, string valued capabilities, such as ce (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. 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. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A \E maps to an ESCAPE character, ^x maps to a control-x for any appropriate x, and the sequences \n \r \t \b \f give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. 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 null character in a string capability it must be encoded as \200. The routines which 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.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in termcap and to build up a description gradually, using partial descriptions with ex to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the termcap file to describe it or bugs in ex. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in /etc/termcap. TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

Basic capabilities

The number of columns on each line for the terminal is given by the co numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the li 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, then this is given by the cl string capability. If the terminal can backspace, then it should have the bs capability, unless a backspace is accomplished by a character other than ^H (ugh) in which case you should give this character as the bc string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the os capability.

A very important point here is that the local cursor motions encoded in termcap are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the am capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the termcap file usually assumes that this is on, i.e. am.

These capabilities suffice to describe hardcopy and "glasstty" terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a cm string capability, with printf(3s) like escapes %x in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the cm string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the % encodings have the following meanings:

%d	as in printf, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds x to value, then %.

```
%>xy if value > x adds y, no output.
%r   reverses order of line and column, no output
%i   increments line/column (for 1 origin)
%%   gives a single %
%n   exclusive or row and column with 0140 (DM2500)
%B   BCD (16*(x/10)) + (x%10), no output.
%D   Rev coding (x-2*(x%16)), no output (Delta Data)
```

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its cm capability is "cm=6\E&%r%2c%2Y". The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, "cm=^T%.%.". Terminals which use "%." need to be able to backspace the cursor (bs or bc), and to move the cursor up one line on the screen (up introduced below). This is necessary because it is not always safe to transmit \t, \n ^D and \r, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "cm=\E=%+ %+ ".

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as nd (nondestructive space). If it can move the cursor up a line on the screen in the same column, this should be given as up. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as ho; similarly a fast way of getting to the lower left hand corner can be given as ll; this may involve going up with up from the home position, but the editor will never do this itself (unless ll does) because it makes no assumption about the effect of moving up from the home position.

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 ce. If the terminal can clear from the current position to the end of the display, then this should be given as cd. The editor only uses cd from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as al; 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 dl; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as sb, but just al suffices. 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 let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with sb may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using termcap. 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 find out which 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 spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal 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". If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not not falling into one of these two classes.

The editor can handle 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 im the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as ei the sequence to leave insert mode (give this, with an

empty value also if you gave im so). Now give as ic any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give ic, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds 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.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability mi to speed up inserting in this case. Omitting mi will affect only speed. Some terminals (notably Datamedia's) must not have mi because of the way their insert mode works.

Finally, you can specify delete mode by giving dm and ed to enter and exit delete mode, and dc to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode these can be given as so and se respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining - half bright is not usually an acceptable "standout" mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then ug should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as us and ue respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as uc. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 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.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given

as vb; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of ex, this can be given as vs and ve, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as ti and te. 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.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability ul. If overstrikes are erasable with a blank, then this should be indicated by giving eo.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as ks and ke. 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 kl, kr, ku, kd, and kh respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as k0, k1, ..., k9. If these keys have labels other than the default f0 through f9, the labels can be given as l0, l1, ..., l9. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the termcap 2 letter codes can be given in the ko capability, for example, ":ko=cl,ll,sf,sb:", which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The ma entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with kl, kr, ku, kd, and kh. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are h for kl, j for kd, k for ku, l for kr, and H for kh. For example, the

mime would be :ma=^Kj^Zk^Xl: indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as pc.

If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, then this can be given as ta.

Hazeltine terminals, which don't allow '~' characters to be printed should indicate hz. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate nc. Early Concept terminals, which ignore a linefeed immediately after an am wrap, should indicate xn. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), xs should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate xt. Other specific terminal problems may be corrected by adding more capabilities of the form xx.

Other capabilities include is, an initialization string for the terminal, and if, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, is will be printed before if. This is useful where if is /usr/lib/tabset/std but is clears the tabs first.

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 two entries must not exceed 1024. Since termlib routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with xx@ where xx is the capability. For example, the entry

```
hn|2621nl:ks@:ke@:tc=2621:
```

defines a 2621nl that does not have the ks or ke capabilities, and 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.

TERMCAP(5)

FILES

/etc/termcap file containing terminal descriptions

SEE ALSO

ex(1), curses(3), termcap(3), tset(1), vi(1), ul(1), more(1)

AUTHOR

William Joy
Mark Horton added underlining and keypad support

RESTRICTIONS

Ex allows only 256 characters for string capabilities, and the routines in termcap(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The ma, vs, and ve entries are specific to the vi program.

Not all programs support all entries. There are entries that are not supported by any program.

NAME

tp - DEC/mag tape formats

DESCRIPTION

The command tp dumps files to and extracts files from DECtape and magtape. The formats of these tapes are the same except that magtapes have larger directories.

Block zero contains a copy of a stand-alone bootstrap program.

Blocks 1 through 24 for DECtape (1 through 62 for magtape) contain a directory of the tape. There are 192 (resp. 496) entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

```
struct {
        char pathname[32];
        int  mode;
        char uid;
        char gid;
        char unused1;
        char size[3];
        long modtime;
        int  tapeaddr;
        char unused2[16];
        int  checksum;
};
```

The path name entry is the path name of the file when put on the tape. If the pathname starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null byte. Mode, uid, gid, size and time modified are the same as described under i-nodes (see file system filsys(5)). The tape address is the tape block number of the start of the contents of the file. Every file starts on a block boundary. The file occupies (size+511)/512 blocks of continuous tape. The checksum entry has a value such that the sum of the 32 words of the directory entry is zero.

Blocks above 25 (resp. 63) are available for file storage.

A fake entry has a size of zero.

SEE ALSO

filsys(5), tp(1)

RESTRICTIONS

The pathname, uid, gid, and size fields are too small.

TTYS(5)

NAME

ttys - terminal initialization data

DESCRIPTION

The ttys file is read by the init program and specifies which terminal special files are to have a process created for them which will allow people to log in. It contains one line per special file.

The first character of a line is either '0', '1', '2' or '3'; zero causes the line to be ignored, one causes it to be effective as a dialup line, two specifies local terminal operation, and three specifies local terminal operation, but does not run logins on that line.

The second character is used as an argument to getty(8), which performs such tasks as baud-rate recognition, reading the login name, and calling login. Refer to getty(8) for a list of these special characters and their meanings.

The remainder of the line is the terminal's entry in the device directory, /dev.

FILES

/etc/ttys

SEE ALSO

init(8), getty(8), login(1)
ULTRIX-11 System Management Guide, Section 4.7

NAME

ttytype - terminal type data

SYNOPSIS

/etc/ttytype

DESCRIPTION

The ttytype file is read by the login program and specifies a terminal type name and the line name. It contains one line per terminal port.

The first entry is a terminal type name, then a space, and then the name of the tty line, minus /dev/.

Terminal types are of the order - vt100, vt100w, vt52, la36, dialup, la120, etc.

SEE ALSO

login(1)

TYPES(5)

NAME

types - primitive system data types

SYNOPSIS

```
#include <sys/types.h>
```

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
/*
 * SCCSID: @(#)types.h   1.2  8/13/84
 */
#ifndef   __TYPES__
#define   __TYPES__
typedef   long           daddr_t;       /* disk address */
typedef   char *         caddr_t;       /* core address */
typedef   unsigned int   ino_t;         /* i-node number */
typedef   long           time_t;        /* a time */
typedef   int            label_t[7];    /* program status */
typedef   int            dev_t;         /* device code */
typedef   long           off_t;         /* offset in file */

typedef   char      bool_t;         /* Boolean */
typedef   unsigned short comp_t;        /* "floating tp":
                                           3 bits base 8 exp,
                                           13 bits fraction */
typedef   unsigned  memaddr;  /* core or swap address */
typedef   struct {short r[1];}*    physadr;
typedef   unsigned  size_t;         /* size of process segments */
typedef   long      ubadr_t;  /* UNIBUS address */
typedef   unsigned short u_short;
typedef   short          void;          /* So Ritchie C compiler
                                           won't die */

    /* selectors and constructor for device code */
#define   major(x)       (int)(((unsigned)x>>8))
#define   minor(x)       (int)(x&0377)
#define   makedev(x,y)   (dev_t)((x)<<8|(y))
#endif
```

The form daddr_t is used for disk addresses except in an i-node on disk, see filsys(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
filsys(5), time(2), lseek(2), adb(1)

UTMP(5)

NAME

utmp, wtmp - login records

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

The utmp file allows one to discover information about who is currently using UNIX. The file is a sequence of entries with the following structure declared in the include file:

```
/*
 * SCCSID: @(#)utmp.h    1.0  11/12/83
 */
struct utmp {
     char ut_line[8];          /* tty name */
     char ut_name[8];          /* user id */
     long ut_time;        /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of time(2).

The wtmp file records all logins and logouts. Its format is exactly like utmp except that a null user name indicates a logout on the associated terminal. Furthermore, the terminal name '~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names '|' and '}' indicate the system-maintained time just before and just after a date command has changed the system's idea of the time.

Wtmp is maintained by login(1) and init(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by ac(1).

FILES

/etc/utmp
/usr/adm/wtmp

SEE ALSO

login(1), init(8), who(1), ac(1)

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).

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

SYNOPSIS

/usr/games/backgammon

DESCRIPTION

This program does what you expect. It will ask whether you need instructions.

NAME

banner - make long posters

SYNOPSIS

/usr/games/banner

DESCRIPTION

Banner reads the standard input and prints it sideways in huge built-up letters on the standard output.

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'.

? (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 (DEL) and the action and standing will be printed.

NAME

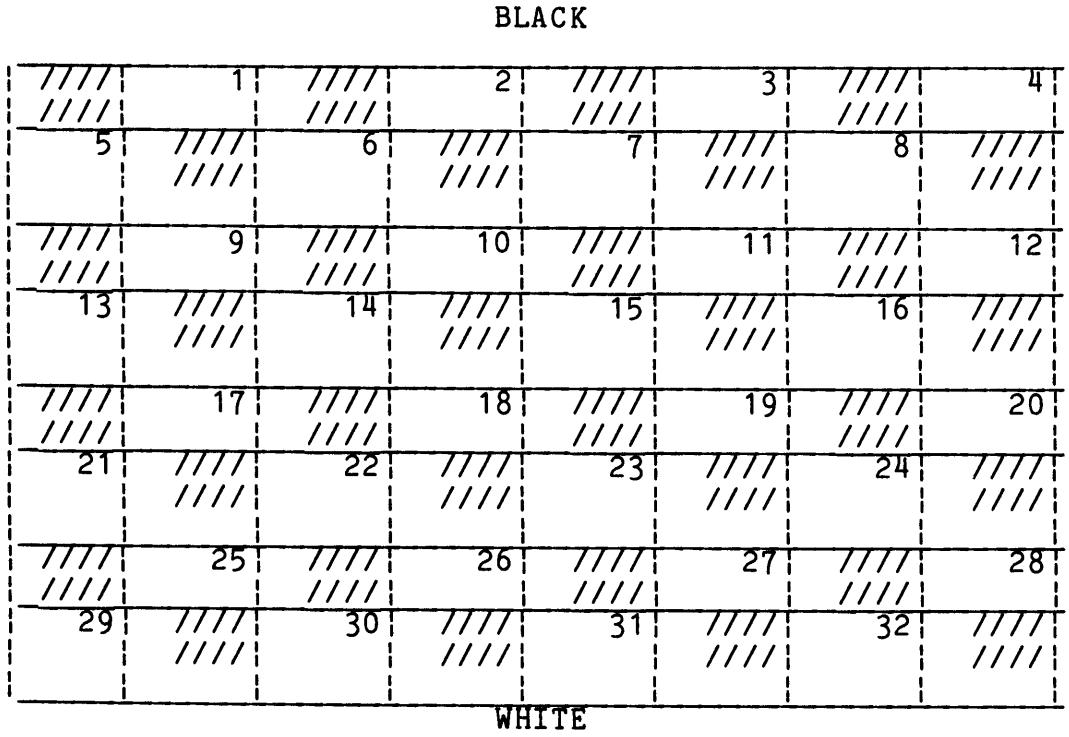
checkers - game

SYNOPSIS

/usr/games/checkers

DESCRIPTION

Checkers uses standard notation for the board:



Black plays first. The program normally plays white. To specify a move, name the square moved from and the square moved to. For multiple jumps name all the squares touched.

Certain commands may be given instead of moves:

reverse Reverse roles; the program takes over your pieces.

backup Undo the last move for each player.

list Print the record of the game.

move Let the program select a move for you.

print Print a map of the present position.

NAME

chess - the game of chess

SYNOPSIS

/usr/games/chess

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.

FILES

/usr/lib/book opening 'book'

DIAGNOSTICS

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

WARNING

Over-use of this program will cause it to go away.

RESTRICTIONS

Pawns may be promoted only to queens.

NAME

ching, fortune - the book of changes and other cookies

SYNOPSIS

/usr/games/ching [hexagram]

/usr/games/fortune

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 (E.g., "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 have the values six or nine, they are moving lines; for each 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, the Unix 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.

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.

RESTRICTIONS
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

ttt, cubic - tic-tac-toe

SYNOPSIS

/usr/games/ttt

/usr/games/cubic

DESCRIPTION

Ttt is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

Cubic plays three-dimensional tic-tac-toe on a 4x4x4 board. Moves are specified as a sequence of three coordinate numbers in the range 1-4.

FILES

/usr/games/ttt.k learning file

MAZE(6)

NAME

maze - generate a maze problem

SYNOPSIS

/usr/games/maze/

DESCRIPTION

Maze asks a few questions and then prints a maze.

RESTRICTIONS

Some mazes (especially small ones) have no solutions.

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

quiz - test your knowledge

SYNOPSIS

/usr/games/quiz [-i file] [-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 '|' 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/*

RESTRICTIONS

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

NAME

reversi - a game of dramatic reversals

SYNOPSIS

/usr/games/reversi [[-r] file]

DESCRIPTION

Reversi (also known as 'friends', 'Chinese friends' and 'Othello') is played on an 8x8 board using two-sided tokens. Each player takes his turn by placing a token with his side up in an empty square. During the first four turns, players may only place tokens in the four central squares of the board. Subsequently, with each turn, a player must capture one or more of his opponent's tokens. He does this by placing one of his tokens such that it and another of his tokens embrace a solid line of his opponent's horizontally, vertically or diagonally. Captured tokens are flipped over and thus can be re-captured. If a player cannot outflank his opponent he forfeits his turn. The play continues until the board is filled or until no more outflanking is possible.

In this game, your tokens are asterisks and the machine's are at-signs. You move by typing in the row and column at which you want to place your token as two digits (1-8), optionally separated by blanks or tabs. You can also type

c to continue the game after hitting break (this is only necessary if you interrupt the machine while it is deliberating).

g n to start reversi playing against itself for the next n moves (or until the break key is hit).

n to stop printing the board after each move.

o to start it up again.

p to print the board regardless.

q to quit (without dishonor).

s to print the score.

Reversi also recognizes several commands which are valid only at the start of the game, before any moves have been made. They are

f to let the machine go first.

h n to ask for a handicap of from one to four corner squares. If you're good, you can give the machine a handicap by typing a negative number.

REVERSI(6)

l n to set the amount of lookahead used by the machine in searching for moves. Zero means none at all. Four is the default. Greater than six means you may fall asleep waiting for the machine to move.

t n to tell reversi that you will only need n seconds to consider each move. If you fail to respond in the allotted time, you forfeit your turn.

If reversi is given a file name as an argument, it will checkpoint the game, move by move, by dumping the board onto file. The -r option will cause reversi to restart the game from file and continue logging.

NAME

ttt, cubic - tic-tac-toe

SYNOPSIS

/usr/games/ttt

/usr/games/cubic

DESCRIPTION

Ttt is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

Cubic plays three-dimensional tic-tac-toe on a 4x4x4 board. Moves are specified as a sequence of three coordinate numbers in the range 1-4.

FILES

/usr/games/ttt.k learning file

NAME

hangman, words - word games

SYNOPSIS

/usr/games/hangman [dict]

/usr/games/words

DESCRIPTION

Hangman chooses a word at least seven letters long from a word list. The user is to guess letters one at a time.

The optional argument names an alternate word list. The special name '-a' gets a particular very large word list.

Words prints all the uncapitalized words in the word list that can be made from letters in string.

FILES

/usr/dict/words the regular word list
/crp/dict/web2 the the -a word list

DIAGNOSTICS

After each round, hangman reports the average number of guesses per round and the number of rounds.

RESTRICTIONS

Hyphenated compounds are run together.

UNIX software is distributed without the -a word list.

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).

RESTRICTIONS

It will never replace Space War.

NAME

ascii - map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

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

FILES

/usr/pub/ascii

EQNCHAR (7)

NAME

eqnchar – special character definitions for eqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] | **troff** [options]

neqn /usr/pub/eqnchar [files] | **nroff** [options]

DESCRIPTION

Eqnchar contains *troff* and *nroff* character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with *eqn* and *neqn*. 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

troff(1), eqn(1)

NAME

greek − graphics for extended TTY-37 type-box

SYNOPSIS

cat /usr/pub/greek [| **greek** −Tterminal]

DESCRIPTION

Greek gives the mapping from ascii to the 'shift out' graphics in effect between SO and SI on model 37 Teletypes with a 128-character type-box. These are the default greek characters produced by *nroff*. The filters of *greek*(1) attempt to print them on various other terminals. The file contains:

alpha	α	A	beta	β	B	gamma	γ	\
GAMMA	Γ	G	delta	δ	D	DELTA	Δ	W
epsilon	ϵ	S	zeta	ζ	Q	eta	η	N
THETA	Θ	T	theta	θ	O	lambda	λ	L
LAMBDA	Λ	E	mu	μ	M	nu	ν	@
xi	ξ	X	pi	π	J	PI	Π	P
rho	ρ	K	sigma	σ	Y	SIGMA	Σ	R
tau	τ	I	phi	ϕ	U	PHI	Φ	F
psi	ψ	V	PSI	Ψ	H	omega	ω	C
OMEGA	Ω	Z	nabla	∇	[	not	$\neg$	_
partial	∂	]	integral	$\int$	^			

SEE ALSO

greek(1)
troff(1)

NAME

hier - file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative directory hierarchy.

```
/    root
/dev/
     devices (4)
     console
          main console, tty(4)
     tty* terminals, tty(4)
     cat  phototypesetter cat(4)
     rp*  disks, rp, hp(4)
     rrp* raw disks, rp, hp(4)
     ...
/bin/
     utility programs, cf /usr/bin/ (1)
     as   assembler first pass, cf /usr/lib/as2
     cc   C compiler executive, cf /usr/lib/c[012]
     ...
/lib/
     object libraries and other stuff, cf /usr/lib/
     libc.a
          system calls, standard I/O, etc. (2,3,3S)
     libm.a
          math routines (3M)
     libplot.a
          plotting routines, plot(3)
     libF77.a
          Fortran runtime support
     libI77.a
          Fortran I/O
     ...
     as2  second pass of as(1)
     c[012]
          passes of cc(1)
     ...
/etc/
     essential data and dangerous maintenance utilities
     passwd
          password file, passwd(5)
     group
          group file, group(5)
     motd message of the day, login(1)
     mtab mounted file table, mtab(5)
     ddate
          dump history, dump(1)
     ttys properties of terminals, ttys(5)
     getty
          part of login, getty(8)
```

```
        init the father of all processes, init(8)
        rc   shell program to bring the system up
        cron the clock daemon, cron(8)
        mount
             mount(1)
        wall wall(1)
        ...
   /tmp/
        temporary files, usually on a fast device, cf /usr/tmp/
        e*   used by ed(1)
        ctm* used by cc(1)
        ...
   /usr/
        general-pupose directory, usually a mounted file system
        adm/ administrative information
             wtmp login history, utmp(5)
             messages
                  hardware error messages
             tracct
                  phototypesetter accounting, troff(1)
             vpacct
                  line printer accounting lpr(1)
   /usr /bin
        utility programs, to keep /bin/ small
        tmp/ temporaries, to keep /tmp/ small
             stm* used by sort(1)
             raster
                  used by plot(1)
        dict/
             word lists, etc.
             words
                  principal word list, used by look(1)
             spellhist
                  history file for spell(1)
        games/
             bj   blackjack
             hangman
             quiz.k/
                  what quiz(6) knows
                  index
                       category index
                  africa
                       countries and capitals
                  ...
             ...
         include/
              standard #include files
              a.out.h
                   object file layout, a.out(5)
              stdio.h
                   standard I/O, stdio(3)
              math.h
```

HIER(7)

```
                (3M)
           ...
           sys/ system-defined layouts, cf /usr/sys/h
                acct.h
                     process accounts, acct(5)
                buf.h
                     internal system buffers
                ...
      lib/ object libraries and stuff, to keep /lib/ small
           lint[12]
                subprocesses for lint(1)
           llib-lc
                dummy declarations for /lib/libc.a, used by
                lint(1)
           llib-lm
                dummy declarations for /lib/libc.m
           atrun
                scheduler for at(1)
           struct/
                passes of struct(1)
           ...
           tmac/
                macros for troff(1)
                tmac.an
                     macros for man(7)
                tmac.s
                     macros for ms(7)
                ...
           font/
                fonts for troff(1)
                R    Times Roman
                B    Times Bold
                ...
           uucp/
                programs and data for uucp(1)
                L.sys
                     remote system names and numbers
                uucico
                     the real copy program
                ...
           suftab
                table of suffixes for hyphenation, used by
                troff(1)
           units
                conversion tables for units(1)
           eign list of English words to be ignored by ptx(1)
/usr/      man/
     volume 1 of this manual, man(1)
           man0/
                general
                intro
                     introduction to volume 1, ms(7) format
```

```
          xx   template for manual page
     man1/
          chapter 1
          as.1
          mount.1m
          ...
     cat1/
          preprinted pages for man1/
          as.1
          mount.1m
     ...
spool/
     delayed execution files
     at/  used by at(1)
     lpd/ used by lpr(1)
          lock present when line printer is active
          cf*  copy of file to be printed, if necessary
          df*  daemon control file, lpd(8)
          tf*  transient control file, while lpr is
               working
     uucp/
          work files and staging area for uucp(1)
          LOGFILE
               summary log
          LOG.*
               log file for one transaction
mail/
     mailboxes for mail(1)
     uid  mail file for user uid
     uid.lock
          lock file while uid is receiving mail
wd   initial working directory of a user, typically wd
     is the user's login name
     .profile
          set environment for sh(1), environ(5)
     calendar
          user's datebook for calendar(1)
doc/ papers, mostly in volume 2 of this manual,
     typically in ms(7) format
     as/  assembler manual
     c    C manual
     ...
sys/ system source
     dev/ device drivers
          bio.c
               common code
          cat.c
               cat(4)
          dh.c DH11, tty(4)
          tty  tty(4)
          ...
     conf/
```

```
                hardware-dependent code
                mch.s
                     assembly language portion
                conf configuration generator
                ...
           h/   header (include) files
                acct.h
                     acct(5)
                stat.h
                     stat(2)
                ...
           sys/ source for system proper
                main.c
                pipe.c
                sysent.c
                     system entry points
                ...
/usr/      src/
     source programs for utilities, etc.
           cmd/ source of commands
                as/  assembler
                     makefile
                          recipe for rebuilding the assembler
                     as1?.s
                          source of pass1
                ar.c source for ar(1)
                ...
                troff/
                     source for nroff and troff(1)
                     nmake
                          makefile for nroff
                     tmake
                          makefile for troff
                     font/
                          source for font tables,
                          /usr/lib/font/
                          ftR.c
                               Roman
                          ...
                     term/
                          terminal characteristics tables,
                          /usr/lib/term/
                          tab300.c
                               DASI 300
                          ...
                     ...
           libc/
                source for functions in /lib/libc.a
                crt/ C runtime support
                     ldiv.s
                          division into a long
                     lmul.s
```

```
                    multiplication to produce long
               ...
          csu/ startup and wrapup routines needed with
               every C program
               crt0.s
                    regular startup
               mcrt0.s
                    modified startup for cc -p
          sys/ system calls (2)
               access.s
               alarm.s
               ...
          stdio/
               standard I/O functions (3S)
               fgets.c
               fopen.c
               ...
          gen/ other functions in (3)
               abs.c
               atof.c
               ...
          compall
               shell procedure to compile libc
          mklib
               shell procedure to make /lib/libc.a
     libI77/
          source for /lib/libI77
     libF77/
     ...
     games/
          source for /usr/games
```

SEE ALSO

ls(1), ncheck(1), find(1), grep(1)

RESTRICTIONS

The position of files is subject to change without notice.

NAME

man - macros to typeset manual

SYNOPSIS

nroff -man file ...

troff -man file ...

DESCRIPTION

These macros are used to lay out pages of this manual. A skeleton page may be found in the file /usr/man/man0/xx.

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 .SM followed by .B 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 troff.

*S Change to default type size.

FILES

/usr/lib/tmac/tmac.an
/usr/man/man0/xx

SEE ALSO

troff(1), man(1)

RESTRICTIONS

Relative indents don't nest.

REQUESTS

Request	Cause Break	If no Argument	Explanation
.B t	no	t=n.t.l.	*Text t is bold.
.BI t	no	t=n.t.l.	Join words of t alternating bold and italic.
.BR t	no	t=n.t.l.	Join words of t alternating bold and Roman.

.DT	no	.5i li...	Restore default tabs.
.HP i	yes	i=p.i.*	Set prevailing indent to i. Begin paragraph with hanging indent.
.I t	no	t=n.t.l.	Text t is italic.
.IB t	no	t=n.t.l.	Join words of t alternating italic and bold.
.IP x i	yes	x=""	Same as .TP with tag x.
.IR t	no	t=n.t.l.	Join words of t alternating italic and Roman.
.LP	yes	-	Same as .PP.
.PD d	no	d=.4v	Interparagraph distance is d.
.PP	yes	-	Begin paragraph. Set prevailing indent to .5i.
.RE	yes	-	End of relative indent. Set prevailing indent to amount of starting .RS.
.RB t	no	t=n.t.l.	Join words of t alternating Roman and bold.
.RI t	no	t=n.t.l.	Join words of t alternating Roman and italic.
.RS i	yes	i=p.i.	Start relative indent, move left margin in distance i. Set prevailing indent to .5i for nested indents.
.SH t	yes	t=n.t.l.	Subhead.
.SM t	no	t=n.t.l.	Text t is small.
.TH n c x	yes	-	Begin page named n of chapter c; x is extra commentary, e.g. 'local', for page foot. Set prevailing indent and tabs to .5i.
.TP i	yes	i=p.i.	Set prevailing indent to i. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

NAME

ms - macros for formatting manuscripts

SYNOPSIS

```
nroff -ms [ options ] file ...
troff -ms [ 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
```

Output of the eqn, neqn, refer, and tbl(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.s

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1)

REQUESTS

Request	Initial Value	Cause Break	Explanation
.1C	yes	yes	One column format on a new page.
.2C	no	yes	Two column format.
.AB	no	yes	Begin abstract.
.AE	-	yes	End abstract.
.AI	no	yes	Author's institution follows. Suppressed in TM.
.AT	no	yes	Print 'Attached' and turn off line filling.
.AU x y	no	yes	Author's name follows. x is location and y is extension, ignored except in TM.
.B x	no	no	Print x in boldface; if no argument switch to boldface.
.B1	no	yes	Begin text to be enclosed in a box.
.B2	no	yes	End text to be boxed . print it.
.BT	date	no	Bottom title, automatically invoked at foot of page. May be redefined.

.BX x	no	no	Print x in a box.
.CS x...	-	yes	Cover sheet info if TM format, suppressed otherwise. Arguments are number of text pages, other pages, total pages, figures, tables, references.
.CT	no	yes	Print 'Copies to' and enter no-fill mode.
.DA x	nroff	no	'Date line' at bottom of page is x. Default is today.
.DE	-	yes	End displayed text. Implies .KE.
.DS x	no	yes	Start of displayed text, to appear verbatim line-by-line. x=I for indented display (default), x=L for left-justified on the page, x=C for centered, x=B for make left-justified block, then center whole block. Implies .KS.
.EG	no	-	Print document in BTL format for 'Engineer's Notes.' Must be first.
.EN	-	yes	Space after equation produced by eqn or neqn.
.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.
.FE	-	yes	End footnote.
.FS	no	no	Start footnote. The note will be moved to the bottom of the page.
.HO	-	no	'Bell Laboratories, Holmdel, New Jersey 07733'.
.I x	no	no	Italicize x; if x missing, italic text follows.
.IH	no	no	'Bell Laboratories, Naperville, Illinois 60540'
.IM	no	no	Print document in BTL format for an internal memorandum. Must be first.
.IP x y	no	yes	Start indented paragraph, with hanging tag x. Indentation is y ens (default 5).
.KE	-	yes	End keep. Put kept text on next page if not enough room.
.KF	no	yes	Start floating keep. If the kept text must be moved to the next page, float later text back to this page.
.KS	no	yes	Start keeping following text.
.LG	no	no	Make letters larger.
.LP	yes	yes	Start left-blocked paragraph.
.MF	-	-	Print document in BTL format for 'Memorandum for File.' Must be first.
.MH	-	no	'Bell Laboratories, Murray Hill, New Jersey 07974'.
.MR	-	-	Print document in BTL format for 'Memorandum for Record.' Must be first.
.ND date	troff	no	Use date supplied (if any) only in

			special BTL format positions; omit from page footer.
.NH n	-	yes	Same as .SH, with section number supplied automatically. Numbers are multilevel, like 1.2.3, where n tells what level is wanted (default is 1).
.NL	yes	no	Make letters normal size.
.OK	-	yes	'Other keywords' for TM cover sheet follow.
.PP	no	yes	Begin paragraph. First line indented.
.PT	pg #	-	Page title, automatically invoked at top of page. May be redefined.
.PY	-	no	'Bell Laboratories, Piscataway, New Jersey 08854'
.QE	-	yes	End quoted (indented and shorter) material.
.QP	-	yes	Begin single paragraph which is indented and shorter.
.QS	-	yes	Begin quoted (indented and shorter) material.
.R	yes	no	Roman text follows.
.RE	-	yes	End relative indent level.
.RP	no	-	Cover sheet and first page for released paper. Must precede other requests.
.RS	-	yes	Start level of relative indentation. Following .IP's are measured from current indentation.
.SG x	no	yes	Insert signature(s) of author(s), ignored except in TM. x is the reference line (initials of author and typist).
.SH	-	yes	Section head follows, font automatically bold.
.SM	no	no	Make letters smaller.
.TA x...	5...	no	Set tabs in ens. Default is 5 10 15 ...
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TL	no	yes	Title follows.
.TM x...	no	-	Print document in BTL technical memorandum format. Arguments are TM number, (quoted list of) case number(s), and file number. Must precede other requests.
.TR x	-	-	Print in BTL technical report format; report number is x. Must be first.
.TS x	-	yes	Begin table; if x is H table has repeated heading.
.UL x	-	no	Underline argument (even in troff).
.UX	-	no	'UNIX'; first time used, add footnote 'UNIX is a trademark of Bell Laboratories.'
.WH	-	no	'Bell Laboratories, Whippany, New Jersey 07981'.

NAME

terminals- conventional names

DESCRIPTION

These names are used by certain commands and are maintained as part of the shell environment (see sh(1),environ(5)).

1620	DIABLO 1620 (and others using HyType II)
1620-12	same, in 12-pitch mode
300	DASI/DTC/GSI 300 (and others using HyType I)
300-12	same, in 12-pitch mode
300s	DASI/DTC 300/S
300s-12	same, in 12-pitch mode
33	TELETYPE(Reg.) Model 33
37	TELETYPE Model 37
40-2	TELETYPE Model 40/2
43	TELETYPE Model 43
450	DASI 450 (same as Diablo 1620)
450-12	same, in 12-pitch mode
450-12-8	same, in 12-pitch, 8 lines/inch mode
735	Texas Instruments TI735 (and TI725)
745	Texas Instruments TI745
dumb	terminals with no special features
hp	Hewlett-Packard HP264? series terminals
4014	Tektronix 4014
tn1200	General Electric TermiNet 1200
tn300	General Electric TermiNet 300
vt05	Digital Equipment Corp. VT05

Commands whose behavior may depend on the terminal accept arguments of the form -Tterm, where term is one of the names given above. If no such argument is present, a command may consult the shell environment for the terminal type.

SEE ALSO

stty(1), tabs(1), plot(1), sh(1), environ(5)
troff(1) for nroff

RESTRICTIONS

The programs that ought to adhere to this nomenclature do so only fitfully.

NAME

boot - startup procedures

DESCRIPTION

Refer to ULTRIX-11 System Management Guide, Chapter 3

FILES

/unix - system code
/mdec/hkuboot - copies of primary bootstraps
/mdec/hpuboot
/mdec/mluboot
/mdec/rauboot
/mdec/rdrxuboot
/mdec/rkuboot
/mdec/rluboot
/mdec/rpuboot
/boot - second stage bootstrap
/boot.bu - backup copy of boot

SEE ALSO

ULTRIX-11 Installation Guide Section 1.3

NAME

crash - what to do when the system crashes

DESCRIPTION

Refer to ULTRIX-11 System Management Guide, Chapter 9

SEE ALSO

ULTRIX-11 System Management Guide, Chapter 8 ULTRIX-11 System Management Guide, Chapter 10

NAME

cron - clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file /usr/lib/crontab. Since cron never exits, it should only be executed once. This is best done by running cron from the initialization process through the file /etc/rc; see init(8).

Crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=monday). Each of these patterns may contain a number in the range above; two numbers separated by a minus meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the Shell at the specified times. A percent character in this field is translated to a new-line 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.

Crontab is examined by cron every minute.

FILES

/usr/lib/crontab

DMESG(8)

NAME

dmesg - collect system diagnostic messages to form error log

SYNOPSIS

/etc/dmesg [-] [corefile [namelist]]

DESCRIPTION

Dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. 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. This is typically used with cron(8) to produce the error log /usr/adm/messages by running the command

```
/etc/dmesg - >> /usr/adm/messages
```

every 10 minutes.

The ULTRIX-11 error logging system logs hardware related errors, dmesg is used to save other types of error messages.

The optional corefile and namelist options can be used for postmortem debugging. If corefile is specified, it will be used instead of /dev/mem; if namelist is also specified, it will be used instead of /unix.

FILES

/usr/adm/messages	error log (conventional location)
/usr/adm/msgbuf	scratch file for memory of - option
/dev/mem	default corefile
/unix	default namelist

SEE ALSO

ULTRIX-11 System Management Guide, Chapter 8 ULTRIX-11 System Management Guide, Chapter 9

RESTRICTIONS

Referring to dmesg as the error log is a misnomer.

The system error message buffer is of small finite size. As dmesg is run only every few minutes, not all error messages are guaranteed to be logged. If this happens, dmesg will print '...' before printing the buffer. This can be construed as a blessing rather than a curse.

Error diagnostics generated immediately before a system crash will never get logged.

NAME

getty - set typewriter mode

SYNOPSIS

/etc/getty [char]

DESCRIPTION

Getty is invoked by init(8) immediately after a typewriter is opened following a dial-up. It reads the user's login name and calls login(1) with the name as argument. While reading the name getty attempts to adapt the system to the speed and type of terminal being used.

Init calls getty with a single character argument taken from the ttys(5) file entry for the terminal line. This argument is used as an index into the gettytab(5) database, to determine the characteristics of the line. If there is no /etc/gettytab a set of system defaults is used.

The user's name is read a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is usually then changed and the greeting message is printed again. Successive 'break' characters cycle through some standard set of speeds.

The user's name is terminated by a new-line or carriage-return character. In the second case CRMOD mode is set (see ioctl(2)).

The name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is nonempty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, login is called with the user's name as argument.

Most of the default actions of getty can be circumvented, or modified, by a suitable gettytab table.

The following arguments from the ttys file are understood.

0 Cycles through 300-1200-150-110 baud. Useful as a default for dialup lines accessed by a variety of terminals.

- Intended for an on-line Teletype model 33, for example an operator's console.

1 Optimized for a 150-baud Teletype model 37.

2 Intended for an on-line 9600-baud terminal, for example

GETTY(8)

a Digital VT100 series terminal.

3 Starts at 1200 baud, cycles to 300 and back. Useful with 212 datasets where most terminals run at 1200 speed.

5 Same as '3' but starts at 300.

4 Useful for on-line console DECwriter (LA36).

6 Intended for an on-line 2400-baud terminal.

7 Intended for an on-line 4800-baud terminal.

a Fixed speed 110-baud terminal.

b Fixed speed 135.4-baud terminal.

d Fixed speed 600-baud terminal.

f Fixed speed 1200-baud terminal.

For additional terminal types and special purpose entries, refer to a listing of the /etc/gettytab table.

FILES

/etc/gettytab

SEE ALSO

gettytab(5), init(8), login(1), ioctl(2), ttys(5)
ULTRIX-11 System Management Guide, Section 4.6

NAME

init, rc - process control initialization

SYNOPSIS

/etc/init
/etc/rc

DESCRIPTION

Init is invoked as the last step of the boot procedure. Generally its role is to create a process for each typewriter on which a user may log in.

When init first is executed the console typewriter /dev/console. is opened for reading and writing and the shell is invoked immediately. This feature is used to bring up a single-user system. If the shell terminates, init comes up multi-user and the process described below is started.

When init comes up multiuser, it invokes a shell, with input taken from the file /etc/rc. This command file performs housekeeping like removing temporary files, mounting file systems, and starting daemons.

Then init reads the file /etc/ttys and forks several times to create a process for each typewriter specified in the file. Each of these processes opens the appropriate typewriter for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input, output and error files. Opening the typewriter will usually involve a delay, since the open is not completed until someone is dialed up and carrier established on the channel. Local typewriters are opened immediately, see ttys(5). Then /etc/getty is called with argument as specified by the second character of the ttys file line. Getty reads the user's name and invokes login(1) to log in the user and execute the shell.

Ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of init, which has been waiting for such an event, wakes up and removes the appropriate entry from the file utmp, which records current users, and makes an entry in /usr/adm/wtmp, which maintains a history of logins and logouts. Then the appropriate typewriter is reopened and getty is reinvoked.

Init catches the hangup signal SIGHUP, which causes init to reread the /etc/ttys file. To bring new terminals on-line or take existing terminals off-line, edit the /etc/ttys file and use 'kill -1 1' to send the hangup signal to init. Only those terminals whose flag character, in the /etc/ttys file,

has been changed will be affected.

FILES

/dev/tty?, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), getty(8)
ULTRIX-11 System Management Guide, Sections 1.2.2, and 3.1

NAME

lpd - line printer daemon

SYNOPSIS

/usr/lib/lpd

DESCRIPTION

Lpd is the daemon for the line printer. Lpd uses the directory /usr/spool/lpd. The file lock in that directory is used to prevent two daemons from becoming active. After the program has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with df. Each such file is submitted as a job. Each line of a job file must begin with a key character to specify what to do with the remainder of the line.

L specifies that the remainder of the line is to be sent as a literal.

B specifies that the rest of the line is a file name.

F is the same as B except a form feed is prepended to the file.

U specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.

M is followed by a user ID; after the job is sent, a message is mailed to the user via the mail(1) command to verify the sending of the job.

Any error encountered will cause the daemon to wait and start over. This means that an improperly constructed df file may cause the same job to be submitted repeatedly.

Lpd is automatically initiated by the line printer command, lpr.

To restart lpd (in the case of hardware or software malfunction), it is necessary to first kill the old daemon (if still alive), and remove the lock file before initiating the new daemon. This is done automatically when the system is brought up, by /etc/rc, in case there were any jobs left in the spooling directory when the system last went down. A new line printer daemon is initiated with the command '/etc/lpdrestart'.

FILES

/usr/spool/lpd/*	spool area for line printer daemon
/etc/passwd	to get the user's name
/dev/lp	line printer device
/etc/lpdrestart	to restart the line printer daemon

SEE ALSO

lpr(1)

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 (i.e. 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 (e.g. ed and crypt(1)). Usually its input and output will be pipes.

SEE ALSO

crypt(1), ed(1)

NAME

shutdown - orderly system shutdown

DESCRIPTION

The shutdown command is obsolete, the operator services program is used for system shutdown. Read the ULTRIX-11 System Management Guide, Chapter 5 for a description of the operator services program.

NAME

ulf - universal lineprinter filter

SYNOPSIS

invokename username

DESCRIPTION

Output is sent through ulf before it is allowed to hit the lineprinter.

Ulf filters out control characters, and buffers up lines for devices like diablos. Accounting records are written in /usr/adm/lpacct by ulf at the time of completion of each job.

Ulf checks to see what name it has been called by so that it will know what entry to look up in /etc/printcap. The account name of the user that was responsible for this invocation of ulf is passed as its one and only argument. This argument is needed to determine who to bill the pages to. Ulf supports diablos and dumb (LA180, LA120) "lineprinters".

Ulf uses a "line structure" that is a linked list of arrays to hold enough characters for one physical line on the lineprinter. The first array contains the characters of the line, and all subsequent arrays hold overprinting characters. Blanks are left as zero's, so that they don't have to be printed, just move the printhead whatever way is easiest.

AUTHOR

Cliff Matthews, University of New Mexico

FILES

/etc/printcap

DIAGNOSTICS

Are sent to wherever /etc/printcap specifies with the lf field. I like to see them go to /dev/null, because they tend to be generated in large quantities, and can fill up a filesystem very quickly. If the software doesn't work, the errors should be dumped to disk and examined. After the source of the trouble has been fixed, it is advisable to have the errors routed back to /dev/null.

SEE ALSO

lpr(1), printcap(5), lpd(8)

NAME

update - periodically update the super block

SYNOPSIS

/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), sync(1), init(8)

RESTRICTIONS

With update running, if the CPU is halted just as the sync is executed, a file system can be damaged. This is partially due to DEC hardware that writes zeros when NPR requests fail. A fix would be to have sync(1) temporarily increment the system time by at least 30 seconds to trigger the execution of update. This would give 30 seconds grace to halt the CPU.