Altos System V™ Series 386
Reference (C)

Document History

EDITION	PART NUMBER	DATE
First Edition	690-22869-001	December 1988
Second Edition	690-22869-002	June 1989

Copyright Notice

Trademarks

The Altos logo, as it appears in this manual, is a registered trademark of Altos Computer Systems.

Altos System V is a trademark of Altos Computer Systems.

CP/M and MP/M are trademarks of Digital Research.

DOCUMENTER'S WORKBENCH is a trademark of AT&T Technologies.

IBM is a registered trademark of International Business Machines Corporation.

LaserJet is a trademark of Hewlett Packard Company.

MS-DOS is a registered trademark of Microsoft Corporation.

UNIX is a registered trademark of AT&T.

WorkNet II is a trademark of Altos Computer Systems.

XENIX is a registered trademark of Microsoft Corporation.

Limitations

Altos Computer Systems reserves the right to make changes to the product described in this manual at any time and without notice. Neither Altos nor its suppliers make any warranty with respect to the accuracy of the information in this manual.

GUIDE TO YOUR ALTOS SYSTEM V™ SERIES 386 DOCUMENTATION

RUN-TIME SYSTEM

Installation

Part numbers: 690-21170-*nnn*
690-21869-*nnn*

- Installation and upgrade
- Set up Multidrop and UPS

Using the AOM™ Menu System

Part number: 690-18055-*nnn*

- Easy-to-use menus to access programs
- Menu Manager to add, update, remove menus

Operations Guide

Part number: 690-21171-*nnn*

- System administration
- Accounting, file systems
- Backups, port setup
- Communications (UUCP)
- Error messages

Reference (C)

Part number: 690-22869-*nnn*

- Commands (C)

Reference (M)

Part number: 690-22870-*nnn*

- Miscellaneous files (M)

User's Guide

Part number: 690-21178-*nnn*
(Not shipped with the Run-time system)

- Basic concepts and tasks
- Vi, ed, mail, awk, sed
- Shells: sh and csh

TEXT PROCESSING SYSTEM

DOCUMENTER'S WORKBENCH™

Part numbers: 690-15843-*nnn*
690-15844-*nnn*

- Mm macros, reference
- Nroff, troff, tbl, eqn

DEVELOPMENT SYSTEM

Set part number: 690-21585-000

Reference (CP, S, F)

- Programming commands (CP)
- System calls, library routines (S)
- File formats (F)

Programmer's Guide

- Make, SCCS
- Lex, yacc
- Signals, system resources, device drivers
- Adb, sdb
- Shared libraries

C Compiler Library and User's Guide

- I/O functions, pipes
- Curses, terminfo
- Assembly routines
- As, cc, COFF, lint, ld
- Error processing
- Character and string processing

C Compiler Language Reference

- Elements of C
- Program structure
- Declarations, expressions
- Statements, functions
- Preprocessor directives

Macro Assembler User's Guide and Reference

- How to use masm
- Error messages
- Type declarations
- Operands, expressions
- Directives, file control
- Instruction summary

To order the *User's Guide* or any of the above manuals, call 408/434-6688, ext. 3004 and give the manual title and part number.

About This Manual

USING THIS MANUAL

This reference alphabetically describes the commands and programs that are on the Altos System V™ Run-time System. Altos System V is based on UNIX® System V Release 3 with enhancements from Altos and Microsoft.

ORGANIZATION

This manual contains the commands, programs, and utilities (C) of the Run-time system.

For miscellaneous utilities and files (M), see the *Reference (M)*.

NOTE

The last section of the manual, "Change Information," summarizes the changes that have been made to the manual since the previous version.

MANUAL CONVENTIONS

The documentation conventions used in this manual are explained on the following page.

Symbol	Description
boldface type	What you type. For example: Type **tar tv**
boldface type	Used for command or parameter names that must be typed as shown. **mail** *user*
italic type	Variables (a value that can change), such as *user*. See the previous example. Also for manual titles, such as *Reference (C)* and *Reference (M)*.
Ctrl-d	Keys you press simultaneously (separated by a hyphen and shown in reverse type). For example: Ctrl-d means you press and hold the Ctrl key and then press the **d** key.
Esc **c**	Keys you press sequentially.
[ ]	Optional items in a syntax statement. If you do not use the optional item, the program selects a default action to carry out.
|	Use only one of the separated items.
...	Repeat preceding argument one or more times.
,,,	Repeat the preceding argument one or more times and separate arguments with a comma.
" "	Terms defined in the text. Quotation marks also indicate text from a source code example.

ADDITIONAL REFERENCE MATERIALS

For more information on your operating system, see the following list of manuals. To order a manual, call (408) 434-6688, ext. 3004 and give the manual title and part number.

Owner's Guide (part number 690-21264-*nnn* or 690-20351-*nnn*) describes how to connect computer components and peripherals, turn on power, and use the diagnostic programs.

Using the AOM Menu System (part number 690-18055-*nnn*) describes how to use the Altos Office Manager (AOM) to install software and manage the operating system.

Altos System V User's Guide (part number 690-21178-*nnn*) (not shipped with the Run-time system) explains basic operating system concepts and programs (e.g., vi, ed, sh, csh, mail, sed, and awk).

Altos System V Series 386 Operations Guide (part number 690-21171-*nnn*) tells how to set up the system for users and peripherals, maintain and back up the system, optimize system performance, and use uucp communications programs. This manual also contains system and LP spooler error messages.

Altos System V Series 386 Reference (M) (part number 690-22870-*nnn*) describes the Altos Run-time system utilities and files.

Altos System V Series 386 Development System Set (part number 690-21585-000) contains reference and tutorial material.

Manuals in this set include:

Altos System V Series 386 C Compiler Library and User's Guide
Altos System V Series 386 C Compiler Language Reference
Altos System V Series 386 Programmer's Guide
Altos System V Series 386 Macro Assembler User's Guide and Reference
Altos System V Series 386 Reference (CP, S, F)

DOCUMENTER'S WORKBENCH (part numbers 690-15843-*nnn* and 690-15844-*nnn*) describes mm, nroff, troff, and typesetting functions and commands.

Permuted Index

The Permuted Index on the following pages contains a listing of programs, utilities, files, etc. in the Altos System V Run-time and Development Systems. These programs are described in the *Altos System V Reference.* Volume 1 of the Reference contains the Run-time system commands (C) and miscellaneous (M) sections. Volume 2 contains the Development system programming commands (CP), system calls and library routines (S), and file formats (F). Entries in each section are in alphabetical order.

NOTE

These programs, utilities, files, etc. are subject to change.

The table that follows contains a description of each section and its location.

Description	Section	Manual
Run-time commands	C	Reference (C)
Miscellaneous -- programs and system files used for system maintenance and to access devices	M	Reference (M)
Programming commands	CP	Reference (CP, S, F
System calls and library routines for C and assembly language programming	S	Reference (CP, S, F
File formats -- programs and system files not defined in the M section	F	Reference (CP, S, F

as(CP)	386 Assembler	as(CP)
l3tol(S) ltol3(S) convert between	3-byte integers and long integers	l3tol(S)
tk(C) paginator for Tektronix	4014	tk(C)
integer and base-64 ASCII string	a64l(S) l64a(S) convert between long	a64l(S)
	abort(S) generate an IOT fault	abort(S)
abs(S) return integer	absolute value	abs(S)
ceil(S) fabs(S) floor, ceiling, and	absolute value functions floor(S)	floor(S)
floor(S) fmod(S) floor, ceiling, and	absolute value functions	floor(S)
	abs(S) return integer absolute value	abs(S)
requests	accept(C) reject(C) allow/prevent print	accept(C)
settime(C) change the	access and modification dates of files	settime(C)
touch(C) update	access and modification times of a file	touch(C)
utime(S) set file	access and modification times	utime(S)
login(C) give you system	access	login(C)
sputl(S) sgetl(S)	access long integer data	sputl(S)
dos(C)	access MS-DOS files	dos(C)
sadp(M) disk	access profiler	sadp(M)
ldfcn(F) common object file	access routines	ldfcn(F)
sdwaitv(S) synchronize shared data	access sdgetv(S)	sdgetv(S)
sdenter(S) sdleave(S) synchronize	access to a shared data segment	sdenter(S)
waitsem(S) nbwaitsem(S) wait and check	access to semaphore resource	waitsem(S)
clock(M) provide	access to the time-of-day chip	clock(M)
getutent(S) utmpname(S) endutent(S)	access utmp file entry getut(S)	getut(S)
getut(S) setutent(S) getutline(S)	access utmp file entry	getut(S)
access(S) determine	accessibility of a file	access(S)
file	access(S) determine accessibility of a	access(S)
csplit(C) split files	according to context	csplit(C)
acct(S) enable or disable process	accounting	acct(S)
acct(M) format of per-process	accounting file	acct(M)
acct(C)	accounting system	acct(C)
	acct(C) accounting system	acct(C)
file	acct(M) format of per-process accounting	acct(M)
accounting	acct(S) enable or disable process	acct(S)
trig(S) sin(S) cos(S) tan(S) asin(S)	acos(S) trigonometric functions	trig(S)
killall(C) kill all	active processes	killall(C)
sar(C) system	activity report package	sar(C)
sar(M) system	activity report package	sar(M)
sact(CP) print current SCCS file edit	activity	sact(CP)
debugger	adb(C) invoke x.out general purpose	adb(C)
add.hd(C)	add an additional hard disk	add.hd(C)
nl(C)	add line numbers to a file	nl(C)
map badblock(C)	add new bad sectors to the bad sector	badblock(C)
lpinit(M)	add new line printers	lpinit(M)
putenv(S) change or	add value to environment	putenv(S)
	add.hd(C) add an additional hard disk	add.hd(C)
add.hd(C) add an	additional hard disk	add.hd(C)
upgrade.hd(C) upgrade an	additional hard disk	upgrade.hd(C)
files	admin(CP) create and administer SCCS	admin(CP)
admin(CP) create and	administer SCCS files	admin(CP)
ua(C) user	administration program	ua(C)
uadmin(S)	administrative control	uadmin(S)
machines	aftp(C) transfer files between Altos	aftp(C)
	aliases(M) mail alias file	aliases(M)
mail alias file	aliashash(M) rebuild data base for	aliashash(M)
alarm(S) set a process	alarm clock	alarm(S)
	alarm(S) set a process alarm clock	alarm(S)
brk(S) sbrk(S) change data segment space	allocation	brk(S)

free(S) realloc(S) fast main memory	allocator malloc(S)	malloc(S)
malloc(S) main memory	allocator	malloc(S)
mallopt(S) calloc(S) fast main memory	allocator malloc(S) mallinfo(S)	malloc(S)
terminal mesg(C)	allow or disallow messages sent to a	mesg(C)
get and set maximum number of users	allowed to log in numusers(S)	numusers(S)
accept(C) reject(C)	allow/prevent print requests	accept(C)
aftp(C) transfer files between	Altos machines	aftp(C)
lex(CP) generate programs for lexical	analysis	lex(CP)
editor output	a.out(F) format of assembler and link	a.out(F)
dc(C)	arbitrary precision calculator	dc(C)
bc(C)	arbitrary-precision arithmetic language	bc(C)
cpio(F) format of cpio	archive	cpio(F)
ar(F)	archive file format	ar(F)
xar(F)	archive file format	xar(F)
the archive header of a member of an	archive file ldahread(S) read	ldahread(S)
tar(C)	archive files	tar(C)
file ldahread(S) read the	archive header of a member of an archive	ldahread(S)
streaming tape	archive(C) save a file system to a	archive(C)
ar(CP) maintain	archives and libraries	ar(CP)
xar(CP) maintain	archives and libraries	xar(CP)
cpio(C) copy file	archives in and out	cpio(C)
ranlib(CP) convert	archives to random libraries	ranlib(CP)
	ar(CP) maintain archives and libraries	ar(CP)
	ar(F) archive file format	ar(F)
varargs(F) handles variable	argument list	varargs(F)
getopt(S) get option letter from	argument vector	getopt(S)
expr(C) evaluate	arguments as an expression	expr(C)
echo(C) echo	arguments	echo(C)
bc(C) arbitrary-precision	arithmetic language	bc(C)
asa(C) interpret	asa carriage control characters	asa(C)
characters	asa(C) interpret asa carriage control	asa(C)
ascii(M) map of the	ASCII character set	ascii(M)
convert between long integer and base-64	ASCII string a64l(S) l64a(S)	a64l(S)
	ascii(M) map of the ASCII character set	ascii(M)
	as(CP) 386 Assembler	as(CP)
time to string ctime(S) tzset(S)	asctime(S) cftime(S) convert date and	ctime(S)
trig(S) sin(S) cos(S) tan(S)	asin(S) acos(S) trigonometric functions	trig(S)
	asktime(C) set the system time of day	asktime(C)
a.out(F) format of	assembler and link editor output	a.out(F)
as(CP) 386	Assembler	as(CP)
masm(CP) invoke the macro	assembler	masm(CP)
assert(S) verify program	assertion	assert(S)
	assert(S) verify program assertion	assert(S)
setbuf(S) setvbuf(S)	assign buffering to a stream	setbuf(S)
trig(S) atan(S)	atan2(S) trigonometric functions	trig(S)
trig(S)	atan(S) atan2(S) trigonometric functions	trig(S)
later time	at(C) batch(C) execute commands at a	at(C)
double-precision number strtod(S)	atof(S) convert string to	strtod(S)
strtol(S) atol(S)	atoi(S) convert string to integer	strtol(S)
integer strtol(S)	atol(S) atoi(S) convert string to	strtol(S)
sdget(S) sdfree(S)	attach and detach a shared data segment	sdget(S)
reboot(C)	automatically reboot the system	reboot(C)
reboot the system	autoreboot(C) automatically	autoreboot(C)
language	awk(C) pattern scanning and processing	awk(C)
wait(C) wait completion of	background processes	wait(C)
finc(M) fast incremental	backup	finc(M)
ckbupscd(M) check file system	backup schedule	ckbupscd(M)

frec(M) recover files from a back-up tape _____ frec(M)
badblock(C) add new bad sectors to the bad sector map _____ badblock(C)
badblock(C) add new bad sectors to the bad sector map _____ badblock(C)
bad sector map badblock(C) add new bad sectors to the __ badblock(C)
banner(C) print large letters _____ banner(C)
164a(S) convert between long integer and base-64 ASCII string a64l(S) _____ a64l(S)
of pathnames basename(C) dirname(C) deliver portions _ basename(C)
time at(C) batch(C) execute commands at a later ____ at(C)
language bc(C) arbitrary-precision arithmetic ____ bc(C)
diff bdiff(C) compare files too large for ____ bdiff(C)
cb(CP) beautify C programs _____ cb(CP)
bessel(S) j0(S) y0(S) Bessel functions _____ bessel(S)
bessel(S) j0(S) y0(S) Bessel functions __ bessel(S)
bfs(C) scan big files _____ bfs(C)
bfs(C) scan big files _____ bfs(C)
fwrite(S) fread(S) binary input/output _____ fwrite(S)
whereis(C) locate source, binary, or manual for program _____ whereis(C)
bsearch(S) binary search of a sorted table _____ bsearch(S)
tfind(S) tdelete(S) twalk(S) manage binary search trees tsearch(S) _____ tsearch(S)
creatsem(S) create a binary semaphore _____ creatsem(S)
reset(C) reset the teletype bit _____ reset(C)
ssp(C) remove consecutive blank lines _____ ssp(C)
sync(S) update super block _____ sync(S)
df(M) report number of free disk blocks and inodes _____ df(M)
sum(C) calculate checksum and count blocks in a file _____ sum(C)
boot(M) boot program _____ boot(M)
mkboot(M) convert object file to bootable object file _____ mkboot(M)
table mkunix(M) make bootable system file with driver symbol _ mkunix(M)
table mkunix(M) make bootable system file with kernel symbol _ mkunix(M)
boot(M) boot program _____ boot(M)
brc(M) system initialization procedure __ brc(M)
shutdown(M) bring system to single-user or shutdown _ shutdown(M)
multiuser(C) singleuser(C) bring system up multi/single-user mode __ multiuser(C)
allocation brk(S) sbrk(S) change data segment space brk(S)
table bsearch(S) binary search of a sorted ____ bsearch(S)
bsh(C) invoke the Business shell _____ bsh(C)
stdio(S) standard buffered input/output package _____ stdio(S)
setbuf(S) setvbuf(S) assign buffering to a stream _____ setbuf(S)
mknod(C) build special files _____ mknod(C)
bsh(C) invoke the Business shell _____ bsh(C)
digest(C) create menu system(s) for the Business shell _____ digest(C)
menus(M) format of Business shell menu system _____ menus(M)
swab(S) swap bytes _____ swab(S)
cc(CP) invoke the C compiler _____ cc(CP)
xcc(CP) invoke the XENIX C compiler _____ xcc(CP)
cflow(CP) generate C flow graph _____ cflow(CP)
cpp(CP) the C Language Preprocessor _____ cpp(CP)
lint(CP) check C language usage and syntax _____ lint(CP)
cxref(CP) generate C program cross-reference _____ cxref(CP)
ctrace(CP) C program debugger _____ ctrace(CP)
cb(CP) beautify C programs _____ cb(CP)
xref(CP) cross-reference C programs _____ xref(CP)
xstr(CP) extract strings from C programs _____ xstr(CP)
list(CP) produce C source listing from COFF file _____ list(CP)
create an error message file from C source mkstr(C) _____ mkstr(C)
create an error message file from C source mkstr(CP) _____ mkstr(CP)
cal(C) print a calendar _____ cal(C)

file sum(C)	calculate checksum and count blocks in a	sum(C)
dc(C) arbitrary precision	calculator	dc(C)
cal(C) print a	calendar	cal(C)
	calendar(C) invoke a reminder service	calendar(C)
cu(C)	call another UNIX system	cu(C)
stat(F) return data by stat system	call	stat(F)
malloc(S) mallinfo(S) mallopt(S)	calloc(S) fast main memory allocator	malloc(S)
intro(S) introduce system	calls, functions, and libraries	intro(S)
line printer lp(C)	cancel(C) send/cancel requests to LP	lp(C)
termcap(M) terminal	capability database	termcap(M)
terminfo(M) terminal	capability database	terminfo(M)
description	captoinfo(M) convert termcap to terminfo	captoinfo(M)
asa(C) interpret asa	carriage control characters	asa(C)
	cat(C) concatenate and display files	cat(C)
	cb(CP) beautify C programs	cb(CP)
gencc(CP) create a front end to the	cc command	gencc(CP)
	cc(CP) invoke the C compiler	cc(CP)
	cd(C) change working directory	cd(C)
SCCS delta	cdc(CP) change the delta commentary of	cdc(CP)
absolute value functions floor(S)	ceil(S) fabs(S) floor, ceiling, and	floor(S)
floor(S) ceil(S) fabs(S) floor,	ceiling, and absolute value functions	floor(S)
floor(S) fmod(S) floor,	ceiling, and absolute value functions	floor(S)
	cflow(CP) generate C flow graph	cflow(CP)
string ctime(S) tzset(S) asctime(S)	cftime(S) convert date and time to	ctime(S)
brk(S) sbrk(S)	change data segment space allocation	brk(S)
passwd(C)	change login password	passwd(C)
chmod(S)	change mode of file	chmod(S)
putenv(S)	change or add value to environment	putenv(S)
chown(S)	change owner and group of a file	chown(S)
chown(C) chgrp(C)	change owner or group ID	chown(C)
directory chmod(C)	change permissions of a file or	chmod(C)
nice(S)	change priority of a process	nice(S)
chroot(S)	change root directory	chroot(S)
chroot(C)	change root directory for command	chroot(C)
swap(C)	change swap device configuration	swap(C)
of files settime(C)	change the access and modification dates	settime(C)
delta cdc(CP)	change the delta commentary of SCCS	cdc(CP)
chsize(S)	change the file size	chsize(S)
delta(CP) make a	change to an SCCS file	delta(CP)
cd(C)	change working directory	cd(C)
chdir(S)	change working directory	chdir(S)
pipe(S) create an interprocess	channel	pipe(S)
ungetc(S) push	character back into input stream	ungetc(S)
cuserid(S) get	character login name of the user	cuserid(S)
getc(S) getw(S) fgetc(S) getchar(S) get	character or word from a stream	getc(S)
putc(S) putchar(S) putw(S) fputc(S) put	character or word on a stream	putc(S)
ascii(M) map of the ASCII	character set	ascii(M)
fgrep(C) search a file for a	character string	fgrep(C)
asa(C) interpret asa carriage control	characters	asa(C)
toascii(S) tolower(S) translate	characters conv(S) toupper(S)	conv(S)
islower(S) iscntrl(S) classify	characters ctype(S) isalpha(S)	ctype(S)
ispunct(S) isascii(S) classify	characters ctype(S) isdigit(S)	ctype(S)
tr(C) translate	characters	tr(C)
wc(C) count lines, words, and	characters	wc(C)
	chdir(S) change working directory	chdir(S)
waitsem(S) nbwaitsem(S) wait and	check access to semaphore resource	waitsem(S)
fsck(C) dfsck(C)	check and repair file systems	fsck(C)

lint(CP) check C language usage and syntax _____ lint(CP)
ckbupscd(M) check file system backup schedule _____ ckbupscd(M)
pwck(M) grpck(M) check password/group file _____ pwck(M)
permissions file uucheck(M) check the uucp directories and _____ uucheck(M)
rdchk(S) check to see if there is data to be read rdchk(S)
labelit(M) copy file system with label checking volcopy(M) _____ volcopy(M)
by fsck checklist(M) list file systems processed checklist(M)
sum(C) calculate checksum and count blocks in a file _____ sum(C)
chown(C) chgrp(C) change owner or group ID _____ chown(C)
times(S) get process and child process times _____ times(S)
wait(S) wait for child process to stop or terminate _____ wait(S)
provide access to the time-of-day chip clock(M) _____ clock(M)
libraries chkshlib(CP) tool for comparing shared __ chkshlib(CP)
directory chmod(C) change permissions of a file or chmod(C)
chmod(S) change mode of file _____ chmod(S)
ID chown(C) chgrp(C) change owner or group _ chown(C)
file chown(S) change owner and group of a ____ chown(S)
command chroot(C) change root directory for _____ chroot(C)
chroot(S) change root directory _____ chroot(S)
chsize(S) change the file size _____ chsize(S)
schedule ckbupscd(M) check file system backup ____ ckbupscd(M)
isalpha(S) islower(S) iscntrl(S) classify characters ctype(S) _____ ctype(S)
isdigit(S) ispunct(S) isascii(S) classify characters ctype(S) _____ ctype(S)
inir(M) clean the file system and executes init _ inir(M)
strclean(M) STREAMS error logger cleanup program _____ strclean(M)
uucleanup(M) uucp spool directory cleanup _____ uucleanup(M)
clri(M) clear inode _____ clri(M)
clear(C) clear terminal screen _____ clear(C)
clear(C) clear terminal screen _____ clear(C)
inquiries ferror(S) fileno(S) clearerr(S) feof(S) stream status _____ ferror(S)
csh(C) shell command interpreter with C-like syntax _____ csh(C)
alarm(S) set a process alarm clock _____ alarm(S)
time-of-day chip clock(M) provide access to the _____ clock(M)
clock(S) report CPU time used _____ clock(S)
STREAMS driver clone(M) open any minor device on _____ clone(M)
ldclose(S) ldaclose(S) close a COFF file _____ ldclose(S)
close(S) close a file descriptor _____ close(S)
fclose(S) fflush(S) close or flush a stream _____ fclose(S)
haltsys(C) close the file systems and halt the CPU _ haltsys(C)
directory operations directory(S) closedir(S) rewinddir(S) seekdir(S) ____ directory(S)
close(S) close a file descriptor _____ close(S)
clri(M) clear inode _____ clri(M)
cmp(C) compare two files _____ cmp(C)
dis(CP) object code disassembler _____ dis(CP)
ldclose(S) ldaclose(S) close a COFF file _____ ldclose(S)
ldfhread(S) read the file header of a COFF file _____ ldfhread(S)
list(CP) produce C source listing from COFF file _____ list(CP)
to line number entries of a section of a COFF file ldlseek(S) seek _____ ldlseek(S)
to relocation entries of a section of a COFF file ldrseek(S) seek _____ ldrseek(S)
an indexed/named section header of a COFF file ldshread(S) read _____ ldshread(S)
the index of a symbol table entry of a COFF file ldtbindex(S) compute _____ ldtbindex(S)
read an indexed symbol table entry of a COFF file ldtbread(S) _____ ldtbread(S)
seek to the symbol table of a COFF file ldtbseek(S) _____ ldtbseek(S)
remove symbols and line numbers from COFF file strip(CP) _____ strip(CP)
convert an object file from OMF to COFF fixobj(CP) _____ fixobj(CP)
manipulate line number entries of a COFF function ldlread(S) ldlitem(S) _____ ldlread(S)
ldgetname(S) retrieve symbol name for COFF symbol table entry _____ ldgetname(S)

comb(CP) combine SCCS deltas _____ comb(CP)
comb(CP) combine SCCS deltas _____ comb(CP)
nice(C) run a command at a different priority _____ nice(C)
chroot(C) change root directory for command _____ chroot(C)
env(C) set environment for command execution _____ env(C)
gencc(CP) create a front end to the cc command _____ gencc(CP)
nohup(C) run a command immune to hangups and quits _____ nohup(C)
setpgrp(C) execute command in a new process group _____ setpgrp(C)
sh(C) rsh(C) invoke the shell command interpreter _____ sh(C)
csh(C) shell command interpreter with C-like syntax __ csh(C)
uux(C) execute command on remote UNIX _____ uux(C)
getopt(C) parse command options _____ getopt(C)
uuxqt(M) execute remote command requests _____ uuxqt(M)
system(S) issue a shell command _____ system(S)
time(C) time a command _____ time(C)
at(C) batch(C) execute commands at a later time _____ at(C)
cron(C) execute commands at specified times _____ cron(C)
rc2(M) commands for multi-user environment _____ rc2(M)
install(M) install commands _____ install(M)
intro(C) introduce commands _____ intro(C)
intro(CP) introduce software development commands _____ intro(CP)
rc0(M) commands to stop the operating system ___ rc0(M)
xargs(C) construct and execute commands _____ xargs(C)
two sorted files comm(C) select/reject lines common to ___ comm(C)
mcs(CP) manipulate the object file comment section _____ mcs(CP)
cdc(CP) change the delta commentary of SCCS delta _____ cdc(CP)
ldfcn(F) common object file access routines _____ ldfcn(F)
cprs(CP) compresse a common object file _____ cprs(CP)
ldopen(S) ldaopen(S) open a common object file for reading _____ ldopen(S)
linenum(F) line number entries in a common object file _____ linenum(F)
nm(CP) print name list of common object file _____ nm(CP)
reloc(F) relocation of information for a common object file _____ reloc(F)
scnhdr(F) section header for a common object file _____ scnhdr(F)
syms(F) common object file symbol table format __ syms(F)
conv(CP) convert common object files _____ conv(CP)
filehdr(F) file header for common object files _____ filehdr(F)
size(C) print section sizes of common object files _____ size(C)
seek to the optional file header of a common object ldohseek(S) _____ ldohseek(S)
comm(C) select/reject lines common to two sorted files _____ comm(C)
glossary(C) define common UNIX terms and symbols _____ glossary(C)
ipcs(C) report inter-process communication facilities status _____ ipcs(C)
stdipc(S) ftok(S) standard interprocess communication package _____ stdipc(S)
dircmp(C) compare directories _____ dircmp(C)
sdiff(C) compare files side-by-side _____ sdiff(C)
bdiff(C) compare files too large for diff _____ bdiff(C)
infocmp(M) compare or print terminfo descriptions __ infocmp(M)
diff3(C) compare three files _____ diff3(C)
cmp(C) compare two files _____ cmp(C)
diff(C) compare two text files _____ diff(C)
sccsdiff(CP) compare two versions of an SCCS file ____ sccsdiff(CP)
chkshlib(CP) tool for comparing shared libraries _____ chkshlib(CP)
regcmp(S) compile a regular expression _____ regcmp(S)
regexp(F) regular expression compile and match routines _____ regexp(F)
routines regexp(S) compile regular expression and match ____ regexp(S)
regcmp(CP) compile regular expressions _____ regcmp(CP)
tic(C) compile terminfo source _____ tic(C)
cc(CP) invoke the C compiler _____ cc(CP)

xcc(CP) invoke the XENIX C	compiler	xcc (CP)
yacc(CP) invoke a	compiler-compiler	yacc(CP)
erf(S) erfc(S) error function and	complementary error function	erf(S)
wait(C) wait	completion of background processes	wait(C)
pack(C) pcat(C) unpack(C)	compress and expand files	pack(C)
cprs(CP)	compress a common object file	cprs(CP)
entry of a COFF file ldtbindex(S)	compute the index of a symbol table	ldtbindex(S)
cat(C)	concatenate and display files	cat(C)
ldunix(M)	configurable kernel linker	ldunix(M)
master(M) master	configuration database	master(M)
printers(M) print spooler	configuration file	printers(M)
sysconf(C) get system	configuration information	sysconf(C)
sysconf(S) get system	configuration information	sysconf(S)
pconfig(C) set port	configuration	pconfig(C)
swap(C) change swap device	configuration	swap(C)
shutype(M) UPS shutdown	configuration utility	shutype(M)
lpadmin(M)	configure the LP spooling system	lpadmin(M)
establish an out-going terminal line	connection dial(S)	dial(S)
ssp(C) remove	consecutive blank lines	ssp(C)
system	console display	display(M)
system	console keyboard	keyboard (M)
math(F) math functions and	constants	math(F)
unistd(F) file header for symbolic	constants	unistd(F)
file header for implementation-specific	constants limits(F)	limits(F)
mkfs(M)	construct a file system	mkfs(M)
xargs(C)	construct and execute commands	xargs(C)
uutry(M)	contact remote system with debugging on	uutry(M)
errprint(M) display error log	contents	errprint(M)
recover(C) restore	contents of a file system from tape	recover(C)
dump.hd(C) dump	contents of a hard disk to tape	dump.hd(C)
ls(C) list	contents of directories	ls(C)
csplit(C) split files according to	context	csplit(C)
fcntl(S) file	control	fcntl(S)
uadmin(S) administrative	control	uadmin(S)
uustat(C) uucp status inquiry and job	control	uustat(C)
vc(CP) version	control	vc(CP)
asa(C) interpret asa carriage	control characters	asa(C)
ioctl(S)	control device	ioctl(S)
IEEE floating point environment	control fpgetround(S) fpgetmask(S)	fpgetround(S)
IEEE floating point environment	control fpgetround(S) fpgetsticky(S)	fpgetround(S)
IEEE floating point environment	control fpgetround(S) fpsetmask(S)	fpgetround(S)
IEEE floating point environment	control fpgetround(S) fpsetround(S)	fpgetround(S)
IEEE floating point environment	control fpgetround(S) fpsetsticky(S)	fpgetround(S)
init(M) process	control initialization	init(M)
msgctl(S) message	control operations	msgctl(S)
semctl(S) semaphore	control operations	semctl(S)
shmctl(S) shared memory	control operations	shmctl(S)
fcntl(F) file	control options	fcntl(F)
	conv(CP) convert common object files	conv(CP)
term(M)	conventional names for terminals	term(M)
fixobj(CP)	convert an object file from OMF to COFF	fixobj(CP)
dd(C)	convert and copy a file	dd(C)
ranlib(CP)	convert archives to random libraries	ranlib(CP)
integers l3tol(S) ltol3(S)	convert between 3-byte integers and long	l3tol(S)
ASCII string a64l(S) l64a(S)	convert between long integer and base-64	a64l(S)
conv(CP)	convert common object files	conv(CP)
ctime(S) gmtime(S) localtime(S)	convert date and time to string	ctime(S)

ctime(S) tzset(S) asctime(S) cftime(S) convert date and time to string ctime(S)
ecvt(S) convert floating-point number to string ecvt(S)
scanf(S) fscanf(S) sscanf(S) convert formatted input scanf(S)
file mkboot(M) convert object file to bootable object mkboot(M)
FORTRAN ratfor(CP) convert rational FORTRAN to standard ratfor(CP)
number strtod(S) atof(S) convert string to double-precision strtod(S)
strtol(S) atol(S) atoi(S) convert string to integer strtol(S)
captoinfo(M) convert termcap to terminfo description captoinfo(M)
units(C) convert units units(C)
translate characters conv(S) toupper(S) toascii(S) tolower(S) conv(S)
dd(C) convert and copy a file dd(C)
fcopy(C) copy a floppy diskette fcopy(C)
cpio(C) copy file archives in and out cpio(C)
volcopy(M) labelit(M) copy file system with label checking volcopy(M)
cp(C) copy files cp(C)
uucp(C) uulog(C) uuname(C) copy files from UNIX to UNIX uucp(C)
copy(C) copy groups of files copy(C)
tra(C) copy out a file as it grows tra(C)
public UNIX-to-UNIX system file copy uuto(C) uupick(C) uuto(C)
copy(C) copy groups of files copy(C)
core(F) format of core image file core(F)
core(F) format of core image file core(F)
sinh(S) cosh(S) tanh(S) hyperbolic functions sinh(S)
trigonometric functions trig(S) sin(S) cos(S) tan(S) asin(S) acos(S) trig(S)
sum(C) calculate checksum and count blocks in a file sum(C)
wc(C) count lines, words, and characters wc(C)
cp(C) copy files cp(C)
cpio(F) format of cpio archive cpio(F)
cpio(C) copy file archives in and out cpio(C)
cpio(F) format of cpio archive cpio(F)
cpp(CP) the C Language Preprocessor cpp(CP)
cprs(CP) compresse a common object file cprs(CP)
cpset(C) install utilities cpset(C)
close the file systems and halt the CPU haltsys(C) haltsys(C)
clock(S) report CPU time used clock(S)
creatsem(S) create a binary semaphore creatsem(S)
gencc(CP) create a front end to the cc command gencc(CP)
tmpnam(S) tempnam(S) create a name for a temporary file tmpnam(S)
one creat(S) create a new file or rewrite an existing creat(S)
fork(S) create a new process fork(S)
mkshlib(CP) create a shared library mkshlib(CP)
ctags(C) create a tags file ctags(C)
tee(C) create a tee in a pipe tee(C)
tmpfile(S) create a temporary file tmpfile(S)
source mkstr(C) create an error message file from C mkstr(C)
source mkstr(CP) create an error message file from C mkstr(CP)
pipe(S) create an interprocess channel pipe(S)
admin(CP) create and administer SCCS files admin(CP)
Shell digest(C) create menu system(s) for the Business digest(C)
makedevs(M) create special device files makedevs(M)
makettys(M) create tty special files makettys(M)
umask(S) set and get file creation mask umask(S)
existing one creat(S) create a new file or rewrite an creat(S)
creatsem(S) create a binary semaphore creatsem(S)
cref(CP) make a cross-reference listing cref(CP)
times cron(C) execute commands at specified cron(C)
crontab(C) manage user crontab files crontab(C)

	crontab(C) manage user crontab files	crontab(C)
xref(CP)	cross-reference C programs	xref(CP)
cxref(CP) generate C program	cross-reference	cxref(CP)
cref(CP) make a	cross-reference listing	cref(CP)
functions	crypt(S) password and file encryption	crypt(S)
C-like syntax	csh(C) shell command interpreter with	csh(C)
context	csplit(C) split files according to	csplit(C)
	ctags(C) create a tags file	ctags(C)
	ct(C) spawn getty to a remote terminal	ct(C)
terminal	ctermid(S) generate file name for	ctermid(S)
date and time to string	ctime(S) gmtime(S) localtime(S) convert	ctime(S)
convert date and time to string	ctime(S) tzset(S) asctime(S) cftime(S)	ctime(S)
	ctrace(CP) C program debugger	ctrace(CP)
iscntrl(S) classify characters	ctype(S) isalpha(S) islower(S)	ctype(S)
isascii(S) classify characters	ctype(S) isdigit(S) ispunct(S)	ctype(S)
	cu(C) call another UNIX system	cu(C)
tty(C) get the	current port name	tty(C)
sact(CP) print	current SCCS file edit activity	sact(CP)
uname(C) print the	current UNIX information	uname(C)
uname(S) get name of	current UNIX system	uname(S)
whoami(C) print effective	current user id	whoami(C)
find the slot in the utmp file of the	current user ttyslot(S)	ttyslot(S)
getcwd(S) get path name of	current working directory	getcwd(S)
scr_dump(F) format of	curses screen image file	scr_dump(F)
optimization package	curses(S) terminal screen handling and	curses(S)
spline(C) interpolate smooth	curves	spline(C)
the user	cuserid(S) get character login name of	cuserid(S)
cross-reference	cxref(CP) generate C program	cxref(CP)
lpd(M) line printer	daemon	lpd(M)
strerr(M) STREAMS error logger	daemon	strerr(M)
xpd(M) transparent printer	daemon	xpd(M)
sdgetv(S) sdwaitv(S) synchronize shared	data access	sdgetv(S)
turn on/off	data collector	sadcon(M)
stat(F) return	data by stat system call	stat(F)
plock(S) lock process, text, or	data in memory	plock(S)
prof(CP) display profile	data	prof(CP)
execseg(S) make a	data region executable	execseg(S)
synchronize access to a shared	data segment sdenter(S) sdleave(S)	sdenter(S)
sdfree(S) attach and detach a shared	data segment sdget(S)	sdget(S)
brk(S) sbrk(S) change	data segment space allocation	brk(S)
sputl(S) sgetl(S) access long integer	data	sputl(S)
rdchk(S) check to see if there is	data to be read	rdchk(S)
types(F) primitive system	data types	types(F)
query terminfo	database	tput(C)
dbminit(S) fetch(S) nextkey(S) perform	database functions dbm(S)	dbm(S)
firstkey(S) store(S) fetch(S) perform	database functions dbm(S)	dbm(S)
master(M) master configuration	database	master(M)
termcap(M) terminal capability	database	termcap(M)
terminfo(M) terminal capability	database	terminfo(M)
ctime(S) gmtime(S) localtime(S) convert	date and time to string	ctime(S)
tzset(S) asctime(S) cftime(S) convert	date and time to string ctime(S)	ctime(S)
date(C) print and set the	date	date(C)
	date(C) print and set the date	date(C)
change the access and modification	dates of files settime(C)	settime(C)
database functions dbm(S)	dbminit(S) fetch(S) nextkey(S) perform	dbm(S)
perform database functions	dbm(S) dbminit(S) fetch(S) nextkey(S)	dbm(S)
perform database functions	dbm(S) firstkey(S) store(S) fetch(S)	dbm(S)

	dc(C) arbitrary precision calculator	dc(C)
	dd(C) convert and copy a file	dd(C)
adb(C) invoke x.out general purpose	debugger	adb(C)
ctrace(CP) C program	debugger	ctrace(CP)
fsdb(M) file system	debugger	fsdb(M)
sdb(C) symbolic	debugger	sdb(C)
uutry(M) contact remote system with	debugging on	uutry(M)
default(M)	default program information directory	default(M)
timezone(M) set	default system time zone	timezone(M)
directory	default(M) default program information	default(M)
glossary(C)	define common UNIX terms and symbols	glossary(C)
sysdef(M) output system	definition	sysdef(M)
basename(C) dirname(C)	deliver portions of pathnames	basename(C)
tail(C)	deliver the last part of a file	tail(C)
change the delta commentary of SCCS	delta cdc(CP)	cdc(CP)
cdc(CP) change the	delta commentary of SCCS delta	cdc(CP)
rmdel(CP) remove a	delta from an SCCS file	rmdel(CP)
	delta(CP) make a change to an SCCS file	delta(CP)
comb(CP) combine SCCS	deltas	comb(CP)
errstop(C) terminate error-logging	demon	errstop(C)
captoinfo(M) convert termcap to terminfo	description	captoinfo(M)
infocmp(M) compare or print terminfo	descriptions	infocmp(M)
close(S) close a file	descriptor	close(S)
dup(S) dup2(S) duplicate an open file	descriptor	dup(S)
sdget(S) sdfree(S) attach and	detach a shared data segment	sdget(S)
access(S)	determine accessibility of a file	access(S)
dtype(C)	determine disk type	dtype(C)
file(C)	determine file type	file(C)
fstyp(M)	determine the file system identifier	fstyp(M)
drive sizefs(C)	determine the size of a logical disk	sizefs(C)
whodo(M)	determine who is doing what	whodo(M)
intro(CP) introduce software	development commands	intro(CP)
swap(C) change swap	device configuration	swap(C)
makedevs(M) create special	device files	makedevs(M)
fold long lines for finite width output	device fold(C)	fold(C)
devinfo(C) display	device information	devinfo(C)
ioctl(S) control	device	ioctl(S)
devnm(C) identify	device name on which files reside	devnm(C)
clone(M) open any minor	device on STREAMS driver	clone(M)
	devinfo(C) display device information	devinfo(C)
files reside	devnm(C) identify device name on which	devnm(C)
and inodes	df(M) report number of free disk blocks	df(M)
fsck(C)	dfsck(C) check and repair file systems	fsck(C)
line connection	dial(S) establish an out-going terminal	dial(S)
bdiff(C) compare files too large for	diff	bdiff(C)
	diff3(C) compare three files	diff3(C)
	diff(C) compare two text files	diff(C)
nice(C) run a command at a	different priority	nice(C)
Business Shell	digest(C) create menu system(s) for the	digest(C)
	dircmp(C) compare directories	dircmp(C)
uucheck(M) check the uucp	directories and permissions file	uucheck(M)
dircmp(C) compare	directories	dircmp(C)
fleece(C) look for files in home	directories	fleece(C)
unlink(M) link and unlink files and	directories link(M)	link(M)
ls(C) list contents of	directories	ls(C)
mv(C) move (rename) files and	directories	mv(C)
rm(C) rmdir(C) remove files or	directories	rm(C)

cd(C) change working directory _____ cd(C)
chdir(S) change working directory _____ chdir(S)
chmod(C) change permissions of a file or directory _____ chmod(C)
chroot(S) change root directory _____ chroot(S)
uucleanup(M) uucp spool directory cleanup _____ uucleanup(M)
default(M) default program information directory _____ default(M)
dir(M) format of a directory _____ dir(M)
getdents(S) read directory entries and put in a file _____ getdents(S)
dirent(F) file system independent directory entry _____ dirent(F)
unlink(S) remove directory entry _____ unlink(S)
chroot(C) change root directory for command _____ chroot(C)
get path name of current working directory getcwd(S) _____ getcwd(S)
mkdir(C) make a directory _____ mkdir(C)
mkdir(S) make a directory _____ mkdir(S)
pwd(C) print working directory name _____ pwd(C)
closedir(S) rewinddir(S) seekdir(S) directory operations directory(S) _____ directory(S)
telldir(S) readdir(S) opendir(S) directory operations directory(S) _____ directory(S)
mknod(S) make a directory, or a special or ordinary file mknod(S)
rmdir(S) remove a directory _____ rmdir(S)
seekdir(S) directory operations directory(S) closedir(S) rewinddir(S) ___ directory(S)
opendir(S) directory operations directory(S) telldir(S) readdir(S) _____ directory(S)
directory entry dirent(F) file system independent _____ dirent(F)
dir(M) format of a directory _____ dir(M)
basename(C) dirname(C) deliver portions of pathnames basename(C)
disable(C) disable logins on a port _____ disable(C)
acct(S) enable or disable process accounting _____ acct(S)
disable(C) disable logins on a port _____ disable(C)
mesg(C) allow or disallow messages sent to a terminal ____ mesg(C)
dis(CP) object code disassembler _____ dis(CP)
set terminal type, modes, speed, line discipline uugetty(M) _____ uugetty(M)
dis(CP) object code disassembler _____ dis(CP)
add.hd(C) add an additional hard disk _____ add.hd(C)
df(M) report number of free disk blocks and inodes _____ df(M)
determine the size of a logical disk drive sizefs(C) _____ sizefs(C)
restore.hd(C) restore a hard disk from tape _____ restore.hd(C)
options(M) floppy disk installation menu _____ options(M)
layout(M) manage hard disk partitions _____ layout(M)
maintain disk partitions _____ fdisk(C)
dump.hd(C) dump contents of a hard disk to tape _____ dump.hd(C)
dtype(C) determine disk type _____ dtype(C)
upgrade.hd(C) upgrade an additional hard disk _____ upgrade.hd(C)
du(C) summarize disk usage _____ du(C)
fcopy(C) copy a floppy diskette _____ fcopy(C)
format(C) format a floppy diskette _____ format(C)
system console display _____ display(M)
see(C) display a file _____ see(C)
devinfo(C) display device information _____ devinfo(C)
vi(C) invoke a screen-oriented display editor _____ vi(C)
errprint(M) display error log contents _____ errprint(M)
cat(C) concatenate and display files _____ cat(C)
hd(C) display files in hexadecimal format _____ hd(C)
od(C) display files in octal format _____ od(C)
prof(CP) display profile data _____ prof(CP)
set up terminal to print screen display pscreen(C) _____ pscreen(C)
hdr(C) display selected parts of an object file hdr(C)
who(C) display who is on the system _____ who(C)
hypot(S) Euclidean distance function _____ hypot(S)

whodo(M) determine who is — doing what — whodo(M)
— dos(C) access MS-DOS files — dos(C)
UNIX — dos disk partitions — fdisk(C)
strtod(S) atof(S) convert string to — double-precision number — strtod(S)
pseudo-random numbers — drand48(S) erand48(S) generate — drand48(S)
lrand48(S) generate pseudo-random/ — drand48(S) mrand48(S) nrand48(S) — drand48(S)
jrand48(S) generate pseudo-random/ — drand48(S) seed48(S) srand48(S) — drand48(S)
graph(C) — draw a graph — graph(C)
manufacturing drive(C) — drive information written during — drive(C)
determine the size of a logical disk — drive sizefs(C) — sizefs(C)
utility program for a streaming tape — drive tapeutil(C) — tapeutil(C)
during manufacturing — drive(C) drive information written — drive(C)
open any minor device on STREAMS — driver clone(M) — clone(M)
mkunix(M) make bootable system file with — driver symbol table — mkunix(M)
— dtype(C) determine disk type — dtype(C)
— du(C) summarize disk usage — du(C)
dump.hd(C) — dump contents of a hard disk to tape — dump.hd(C)
dump(CP) — dump selected parts of an object file — dump(CP)
object file — dump(CP) dump selected parts of an — dump(CP)
to tape — dump.hd(C) dump contents of a hard disk — dump.hd(C)
descriptor dup(S) — dup2(S) duplicate an open file — dup(S)
dup(S) dup2(S) — duplicate an open file descriptor — dup(S)
descriptor — dup(S) dup2(S) duplicate an open file — dup(S)
drive(C) drive information written — during manufacturing — drive(C)
echo(C) — echo arguments — echo(C)
— echo(C) echo arguments — echo(C)
string — ecvt(S) convert floating-point number to — ecvt(S)
ed(C) red(C) invoke the — ed text editor — ed(C)
program end(S) — edata(S) etext(S) last locations in — end(S)
— ed(C) red(C) invoke the ed text editor — ed(C)
sact(CP) print current SCCS file — edit activity — sact(CP)
edit(C) invoke the — edit text editor — edit(C)
— edit(C) invoke the edit text editor — edit(C)
ed(C) red(C) invoke the ed text — editor — ed(C)
edit(C) invoke the edit text — editor — edit(C)
ex(C) invoke a text — editor — ex(C)
ld(CP) invoke the link — editor — ld(CP)
a.out(F) format of assembler and link — editor output — a.out(F)
sed(C) invoke the stream — editor — sed(C)
vi(C) invoke a screen-oriented display — editor — vi(C)
xld(CP) invoke the link — editor — xld(CP)
whoami(C) print — effective current user id — whoami(C)
full regular expression — egrep(C) search file for pattern using — egrep(C)
enable(C) — enable logins on a port — enable(C)
acct(S) — enable or disable process accounting — acct(S)
— enable(C) enable logins on a port — enable(C)
lpenable(C) lpdisable(C) — enable/disable LP line printers — lpenable(C)
crypt(S) password and file — encryption functions — crypt(S)
makekey(M) generate an — encryption key — makekey(M)
gencc(CP) create a front — end to the cc command — gencc(CP)
entry getgrent(S) fgetgrent(S) — endgrent(S) setgrent(S) get group file — getgrent(S)
file entry getpwent(S) fgetpwent(S) — endpwent(S) setpwent(S) get password — getpwent(S)
in program — end(S) edata(S) etext(S) last locations — end(S)
getut(S) getutent(S) utmpname(S) — endutent(S) access utmp file entry — getut(S)
— enroll(C) xsend(C) xget(C) secret mail — enroll(C)
getdents(S) read directory — entries and put in a file — getdents(S)
xlist(S) fxlist(S) get name list — entries from files — xlist(S)

nlist(S) get entries from name list _____ nlist(S)
linenum(F) line number entries in a common object file _____ linenum(F)
ldlitem(S) manipulate line number entries of a COFF function ldlread(S) ___ ldlread(S)
ldlseek(S) seek to line number entries of a section of a COFF file _____ ldlseek(S)
ldrseek(S) seek to relocation entries of a section of a COFF file _____ ldrseek(S)
utmp(M) wtmp(M) format of utmp and wtmp entries _____ utmp(M)
file system independent directory entry dirent(F) _____ dirent(F)
endgrent(S) setgrent(S) get group file entry getgrent(S) fgetgrent(S) _____ getgrent(S)
getgrnam(S) getgrgid(S) get group file entry getgrent(S) _____ getgrent(S)
setpwent(S) get password file entry /fgetpwent(S) endpwent(S) _____ getpwent(S)
getpwuid(S) get password file entry getpwent(S) getpwnam(S) _____ getpwent(S)
utmpname(S) endutent(S) access utmp file entry getut(S) getutent(S) _____ getut(S)
getutline(S) access utmp file entry getut(S) setutent(S) _____ getut(S)
symbol name for COFF symbol table entry ldgetname(S) retrieve _____ ldgetname(S)
compute the index of a symbol table entry of a COFF file ldtbindex(S) _____ ldtbindex(S)
ldtbread(S) read an indexed symbol table entry of a COFF file _____ ldtbread(S)
putpwent(S) write password file entry _____ putpwent(S)
unlink(S) remove directory entry _____ unlink(S)
execution env(C) set environment for command _____ env(C)
environ(M) user environment _____ environ(M)
profile(M) set up environment at login time _____ profile(M)
fpgetmask(S) IEEE floating point environment control fpgetround(S) _____ fpgetround(S)
fpgetsticky(S) IEEE floating point environment control fpgetround(S) _____ fpgetround(S)
fpsetmask(S) IEEE floating point environment control fpgetround(S) _____ fpgetround(S)
fpsetround(S) IEEE floating point environment control fpgetround(S) _____ fpgetround(S)
fpsetsticky(S) IEEE floating point environment control fpgetround(S) _____ fpgetround(S)
environ(M) user environment _____ environ(M)
env(C) set environment for command execution _____ env(C)
getenv(S) return value for environment name _____ getenv(S)
printenv(C) print out the environment _____ printenv(C)
putenv(S) change or add value to environment _____ putenv(S)
rc2(M) commands for multi-user environment _____ rc2(M)
numbers drand48(S) erand48(S) generate pseudo-random _____ drand48(S)
error function erf(S) erfc(S) error function and complementary erf(S)
complementary error function erf(S) erfc(S) error function and _____ erf(S)
sys_nerr(S) sys_errlist(S) errno(S) system error messages _____ sys_nerr(S)
function erf(S) erfc(S) error function and complementary error __ erf(S)
erfc(S) error function and complementary error function erf(S) _____ erf(S)
errprint(M) display error log contents _____ errprint(M)
strclean(M) STREAMS error logger cleanup program _____ strclean(M)
strerr(M) STREAMS error logger daemon _____ strerr(M)
log(M) interface to STREAMS error logging _____ log(M)
mkstr(C) create an error message file from C source _____ mkstr(C)
mkstr(CP) create an error message file from C source _____ mkstr(CP)
perror(S) system error messages _____ perror(S)
sys_errlist(S) errno(S) system error messages sys_nerr(S) _____ sys_nerr(S)
find spelling errors _____ spell(C)
matherr(S) error-handling function _____ matherr(S)
errstop(C) terminate error-logging demon _____ errstop(C)
errprint(M) display error log contents __ errprint(M)
errstop(C) terminate error-logging demon_ errstop(C)
connection dial(S) establish an out-going terminal line ____ dial(S)
setmnt(C) establish /etc/mnttab table _____ setmnt(C)
setmnt(C) establish /etc/mnttab table _____ setmnt(C)
end(S) edata(S) etext(S) last locations in program _____ end(S)
hypot(S) Euclidean distance function _____ hypot(S)
test(C) evaluate an expression _____ test(C)

expr(C) evaluate arguments as an expression ____ expr(C)
ex(C) invoke a text editor ____ ex(C)
file exec(S) execvp(S) execlp(S) execle(S) execv(S) execl(S) execute a ___ exec(S)
execute a file exec(S) execvp(S) execlp(S) execle(S) execv(S) execl(S) ___ exec(S)
execvp(S) execlp(S) execle(S) execv(S) execl(S) execute a file ____ exec(S)
execv(S) execl(S) execute a file exec(S) execvp(S) execlp(S) execle(S) ___ exec(S)
execseg(S) make a data region executable ____ execseg(S)
execseg(S) make a data region executable ____ execseg(S)
execlp(S) execle(S) execv(S) execl(S) execute a file exec(S) execvp(S) ____ exec(S)
regex(S) execute a regular expression ____ regex(S)
setpgrp(C) execute command in a new process group __ setpgrp(C)
uux(C) execute command on remote UNIX ____ uux(C)
at(C) batch(C) execute commands at a later time ____ at(C)
cron(C) execute commands at specified times ____ cron(C)
xargs(C) construct and execute commands ____ xargs(C)
uuxqt(M) execute remote command requests ____ uuxqt(M)
inir(M) clean the file system and executes init ____ inir(M)
env(C) set environment for command execution ____ env(C)
nap(S) suspend execution for a short interval ____ nap(S)
sleep(C) suspend execution for an interval ____ sleep(C)
sleep(S) suspend execution for interval ____ sleep(S)
monitor(S) prepare execution profile ____ monitor(S)
profil(S) execution time profile ____ profil(S)
execl(S) execute a file exec(S) execvp(S) execlp(S) execle(S) execv(S) __ exec(S)
exec(S) execvp(S) execlp(S) execle(S) execv(S) execl(S) execute a ____ exec(S)
creat(S) create a new file or rewrite an existing one ____ creat(S)
false(C) return with a nonzero exit value ____ false(C)
true(C) return with a zero exit value ____ true(C)
exit(S) terminate process ____ exit(S)
pack(C) pcat(C) unpack(C) compress and expand files ____ pack(C)
functions exp(S) pow(S) log(S) exponential, logarithm, and power ____ exp(S)
functions exp(S) sqrt(S) exponential, logarithm, and square root _ exp(S)
expression expr(C) evaluate arguments as an ____ expr(C)
regexp(S) compile regular expression and match routines ____ regexp(S)
regexp(F) regular expression compile and match routines ___ regexp(F)
file for pattern using full regular expression egrep(C) search ____ egrep(C)
expr(C) evaluate arguments as an expression ____ expr(C)
regcmp(S) compile a regular expression ____ regcmp(S)
regex(S) execute a regular expression ____ regex(S)
test(C) evaluate an expression ____ test(C)
regcmp(CP) compile regular expressions ____ regcmp(CP)
logarithm, and power functions exp(S) pow(S) log(S) exponential, ____ exp(S)
and square root functions exp(S) sqrt(S) exponential, logarithm, __ exp(S)
xstr(CP) extract strings from C programs ____ xstr(CP)
value functions floor(S) ceil(S) fabs(S) floor, ceiling, and absolute ____ floor(S)
report inter-process communication facilities status ipcs(C) ____ ipcs(C)
help(C) system help facility ____ help(C)
factor(C) factor a number ____ factor(C)
factor(C) factor a number ____ factor(C)
value false(C) return with a nonzero exit ____ false(C)
ff(M) fast find ____ ff(M)
finc(M) fast incremental backup ____ finc(M)
malloc(S) free(S) realloc(S) fast main memory allocator ____ malloc(S)
mallinfo(S) mallopt(S) calloc(S) fast main memory allocator malloc(S) ____ malloc(S)
abort(S) generate an IOT fault ____ abort(S)
stream fclose(S) fflush(S) close or flush a ____ fclose(S)
fcntl(F) file control options ____ fcntl(F)

	fcntl(S) file control	fcntl(S)
	fcopy(C) copy a floppy diskette	fcopy(C)
UNIX DOS disk partitions	fdisk(C)	fdisk(C)
fopen(S)	fdopen(S) freopen(S) open a stream	fopen(S)
intro(M) introduce miscellaneous	features and files	intro(M)
ferror(S) fileno(S) clearerr(S)	feof(S) stream status inquiries	ferror(S)
stream status inquiries	ferror(S) fileno(S) clearerr(S) feof(S)	ferror(S)
functions dbm(S) dbminit(S)	fetch(S) nextkey(S) perform database	dbm(S)
dbm(S) firstkey(S) store(S)	fetch(S) perform database functions	dbm(S)
head(C) print the first	few lines of a stream	head(C)
fclose(S)	fflush(S) close or flush a stream	fclose(S)
	ff(M) fast find	ff(M)
word from a stream getc(S) getw(S)	fgetc(S) getchar(S) get character or	getc(S)
group file entry getgrent(S)	fgetgrent(S) endgrent(S) setgrent(S) get	getgrent(S)
password file entry getpwent(S)	fgetpwent(S) endpwent(S) setpwent(S) get	getpwent(S)
gets(S)	fgets(S) get a string from a stream	gets(S)
string	fgrep(C) search a file for a character	fgrep(C)
utime(S) set	file access and modification times	utime(S)
ldfcn(F) common object	file access routines	ldfcn(F)
access(S) determine accessibility of a	file	access(S)
acct(M) format of per-process accounting	file	acct(M)
cpio(C) copy	file archives in and out	cpio(C)
tra(C) copy out a	file as it grows	tra(C)
chmod(S) change mode of	file	chmod(S)
chown(S) change owner and group of a	file	chown(S)
mcs(CP) manipulate the object	file comment section	mcs(CP)
fcntl(S)	file control	fcntl(S)
fcntl(F)	file control options	fcntl(F)
uupick(C) public UNIX-to-UNIX system	file copy uuto(C)	uuto(C)
core(F) format of core image	file	core(F)
cprs(CP) compresse a common object	file	cprs(CP)
umask(S) set and get	file creation mask	umask(S)
ctags(C) create a tags	file	ctags(C)
dd(C) convert and copy a	file	dd(C)
delta(CP) make a change to an SCCS	file	delta(CP)
close(S) close a	file descriptor	close(S)
dup(S) dup2(S) duplicate an open	file descriptor	dup(S)
dump selected parts of an object	file dump(CP)	dump(CP)
sact(CP) print current SCCS	file edit activity	sact(CP)
crypt(S) password and	file encryption functions	crypt(S)
endgrent(S) setgrent(S) get group	file entry getgrent(S) fgetgrent(S)	getgrent(S)
getgrnam(S) getgrgid(S) get group	file entry getgrent(S)	getgrent(S)
endpwent(S) setpwent(S) get password	file entry getpwent(S) fgetpwent(S)	getpwent(S)
getpwnam(S) getpwuid(S) get password	file entry getpwent(S)	getpwent(S)
utmpname(S) endutent(S) access utmp	file entry getut(S) getutent(S)	getut(S)
setutent(S) getutline(S) access utmp	file entry getut(S)	getut(S)
putpwent(S) write password	file entry	putpwent(S)
execle(S) execv(S) execl(S) execute a	file exec(S) execvp(S) execlp(S)	exec(S)
fgrep(C) search a	file for a character string	fgrep(C)
grep(C) search a	file for a pattern	grep(C)
expression egrep(C) search	file for pattern using full regular	egrep(C)
ldaopen(S) open a common object	file for reading ldopen(S)	ldopen(S)
ar(F) archive	file format	ar(F)
xar(F) archive	file format	xar(F)
intro(F) introduction to	file formats	intro(F)
mkstr(C) create an error message	file from C source	mkstr(C)
mkstr(CP) create an error message	file from C source	mkstr(CP)

fixobj(CP) convert an object file from OMF to COFF _____ fixobj(CP)
get(CP) get a version of an SCCS file _____ get(CP)
read directory entries and put in a file getdents(S) _____ getdents(S)
group(M) format of the group file _____ group(M)
display selected parts of an object file hdr(C) _____ hdr(C)
filehdr(F) file header for common object files _____ filehdr(F)
constants limits(F) file header for implementation-specific _ limits(F)
unistd(F) file header for symbolic constants _____ unistd(F)
ldfhread(S) read the file header of a COFF file _____ ldfhread(S)
ldohseek(S) seek to the optional file header of a common object _____ ldohseek(S)
split(C) split a file into pieces _____ split(C)
archive header of a member of an archive file ldahread(S) read the _____ ldahread(S)
ldclose(S) ldaclose(S) close a COFF file _____ ldclose(S)
read the file header of a COFF file ldfhread(S) _____ ldfhread(S)
number entries of a section of a COFF file ldlseek(S) seek to line _____ ldlseek(S)
entries of a section of a COFF file ldrseek(S) seek to relocation _____ ldrseek(S)
indexed/named section header of a COFF file ldshread(S) read an _____ ldshread(S)
index of a symbol table entry of a COFF file ldtbindex(S) compute the _____ ldtbindex(S)
an indexed symbol table entry of a COFF file ldtbread(S) read _____ ldtbread(S)
seek to the symbol table of a COFF file ldtbseek(S) _____ ldtbseek(S)
line number entries in a common object file linenum(F) _____ linenum(F)
link(S) link to a file _____ link(S)
produce C source listing from COFF file list(CP) _____ list(CP)
ln(C) make a link to a file _____ ln(C)
mem(M) kmem(M) memory image file _____ mem(M)
convert object file to bootable object file mkboot(M) _____ mkboot(M)
a directory, or a special or ordinary file mknod(S) make _____ mknod(S)
ctermid(S) generate file name for terminal _____ ctermid(S)
mktemp(S) make a unique file name _____ mktemp(S)
nl(C) add line numbers to a file _____ nl(C)
nm(CP) print name list of common object file _____ nm(CP)
null(M) null file _____ null(M)
ttyslot(S) find the slot in the utmp file of the current user _____ ttyslot(S)
more(C) view a file one full screen at a time _____ more(C)
chmod(C) change permissions of a file or directory _____ chmod(C)
fuser(M) identify processes using a file or file structure _____ fuser(M)
creat(S) create a new file or rewrite an existing one _____ creat(S)
passwd(M) password file _____ passwd(M)
for CRTs file perusal filter _____ pg(C)
fseek(S) ftell(S) rewind(S) reposition a file pointer in a stream _____ fseek(S)
lseek(S) move read/write file pointer _____ lseek(S)
printers(M) print spooler configuration file _____ printers(M)
prs(CP) print an SCCS file _____ prs(CP)
pwck(M) grpck(M) check password/group file _____ pwck(M)
read(S) read from file _____ read(S)
locking(S) lock/unlock a file region for read/write _____ locking(S)
of information for a common object file reloc(F) relocation _____ reloc(F)
rev(C) reverse lines of a file _____ rev(C)
rmdel(CP) remove a delta from an SCCS file _____ rmdel(CP)
compare two versions of an SCCS file sccsdiff(CP) _____ sccsdiff(CP)
sccsfile(F) format of an SCCS file _____ sccsfile(F)
section header for a common object file scnhdr(F) _____ scnhdr(F)
format of curses screen image file scr_dump(F) _____ scr_dump(F)
see(C) display a file _____ see(C)
chsize(S) change the file size _____ chsize(S)
stat(S) fstat(S) get file status _____ stat(S)
find the printable strings in an object file strings(C) _____ strings(C)

symbols and line numbers from COFF file strip(CP) remove _____ strip(CP)
identify processes using a file or file structure fuser(M) _____ fuser(M)
mount(C) umount(C) mount/unmount a file structure _____ mount(C)
calculate checksum and count blocks in a file sum(C) _____ sum(C)
syms(F) common object file symbol table format _____ syms(F)
inir(M) clean the file system and executes init _____ inir(M)
ckbupscd(M) check file system backup schedule _____ ckbupscd(M)
fsdb(M) file system debugger _____ fsdb(M)
recover(C) restore contents of a file system from tape _____ recover(C)
fsinfo(M) report information about a file system _____ fsinfo(M)
fstyp(M) determine the file system identifier _____ fstyp(M)
dirent(F) file system independent directory entry _ dirent(F)
statfs(S) fstatfs(S) get file system information _____ statfs(S)
mkfs(M) construct a file system _____ mkfs(M)
mount(S) mount a file system _____ mount(S)
quot(C) summarize file system ownership _____ quot(C)
ustat(S) get file system statistics _____ ustat(S)
fsstat(M) report file system status _____ fsstat(M)
fstab(M) file system table _____ fstab(M)
mnttab(M) mounted file system table _____ mnttab(M)
archive(C) save a file system to a streaming tape _____ archive(C)
sysfs(S) get file system type information _____ sysfs(S)
volcopy(M) labelit(M) copy file system with label checking _____ volcopy(M)
haltsys(C) close the file systems and halt the CPU _____ haltsys(C)
fsck(C) dfsck(C) check and repair file systems _____ fsck(C)
labelit(C) provide labels for file systems _____ labelit(C)
umountall(C) mount/unmount multiple file systems mountall(C) _____ mountall(C)
checklist(M) list file systems processed by fsck _____ checklist(M)
tail(C) deliver the last part of a file _____ tail(C)
tmpfile(S) create a temporary file _____ tmpfile(S)
tempnam(S) create a name for a temporary file tmpnam(S) _____ tmpnam(S)
mkboot(M) convert object file to bootable object file _____ mkboot(M)
tsort(C) sort a file topologically _____ tsort(C)
access and modification times of a file touch(C) update _____ touch(C)
uucico(M) file transport program for uucp system __ uucico(M)
uusched(M) scheduler for the uucp file transport program _____ uusched(M)
ftw(S) walk a file tree _____ ftw(S)
ttys(M) login terminals file _____ ttys(M)
file(C) determine file type _____ file(C)
unget(CP) undo a previous get of an SCCS file _____ unget(CP)
uniq(C) report repeated lines in a file _____ uniq(C)
the uucp directories and permissions file uucheck(M) check _____ uucheck(M)
val(CP) validate an SCCS file _____ val(CP)
mkunix(M) make bootable system file with driver symbol table _____ mkunix(M)
mkunix(M) make bootable system file with kernel symbol table _____ mkunix(M)
write(S) write on a file _____ write(S)
file(C) determine file type _____ file(C)
umask(C) set file-creation mode mask _____ umask(C)
files filehdr(F) file header for common object filehdr(F)
status inquiries ferror(S) fileno(S) clearerr(S) feof(S) stream ____ ferror(S)
csplit(C) split files according to context _____ csplit(C)
admin(CP) create and administer SCCS files _____ admin(CP)
link(M) unlink(M) link and unlink files and directories _____ link(M)
mv(C) move (rename) files and directories _____ mv(C)
aftp(C) transfer files between Altos machines _____ aftp(C)
bfs(C) scan big files _____ bfs(C)
cat(C) concatenate and display files _____ cat(C)

cmp(C) compare two files _____ cmp(C)
select/reject lines common to two sorted files comm(C) _____ comm(C)
conv(CP) convert common object files _____ conv(CP)
copy(C) copy groups of files _____ copy(C)
cp(C) copy files _____ cp(C)
crontab(C) manage user crontab files _____ crontab(C)
diff3(C) compare three files _____ diff3(C)
diff(C) compare two text files _____ diff(C)
dos(C) access MS-DOS files _____ dos(C)
filehdr(F) file header for common object files _____ filehdr(F)
find(C) find files _____ find(C)
hplp(C) hplpR(C) filter files for printing on LaserJet printer __ hplp(C)
frec(M) recover files from a back-up tape _____ frec(M)
uucp(C) uulog(C) uuname(C) copy files from UNIX to UNIX _____ uucp(C)
fspec(F) format specification in text files _____ fspec(F)
fsplit(CP) split ratfor files _____ fsplit(CP)
hd(C) display files in hexadecimal format _____ hd(C)
fleece(C) look for files in home directories _____ fleece(C)
od(C) display files in octal format _____ od(C)
introduce miscellaneous features and files intro(M) _____ intro(M)
lockf(S) record locking on files _____ lockf(S)
makedevs(M) create special device files _____ makedevs(M)
makettys(M) create tty special files _____ makettys(M)
mknod(C) build special files _____ mknod(C)
pr(C) print files on the standard output _____ pr(C)
rm(C) rmdir(C) remove files or directories _____ rm(C)
pcat(C) unpack(C) compress and expand files pack(C) _____ pack(C)
devnm(C) identify device name on which files reside _____ devnm(C)
the access and modification dates of files settime(C) change _____ settime(C)
sdiff(C) compare files side-by-side _____ sdiff(C)
print section sizes of common object files size(C) _____ size(C)
sort(C) sort and merge files _____ sort(C)
tar(C) archive files _____ tar(C)
lpr(C) route named files to printer spooler _____ lpr(C)
bdiff(C) compare files too large for diff _____ bdiff(C)
what(C) identify files _____ what(C)
fxlist(S) get name list entries from files xlist(S) _____ xlist(S)
filesystem(M) format of a system volume _ filesystem(M)
filter file for CRT _____ pg(C)
printer hplp(C) hplpR(C) filter files for printing on LaserJet ___ hplp(C)
finc(M) fast incremental backup _____ finc(M)
ff(M) fast find _____ ff(M)
find(C) find files _____ find(C)
finger(C) find information about users _____ finger(C)
look(C) find lines in a sorted list _____ look(C)
ttyname(S) isatty(S) find name of a terminal _____ ttyname(S)
library lorder(CP) find ordering relation for object _____ lorder(CP)
file strings(C) find the printable strings in an object _ strings(C)
current user ttyslot(S) find the slot in the utmp file of the ___ ttyslot(S)
find(C) find files _____ find(C)
finger(C) find information about users __ finger(C)
fold(C) fold long lines for finite width output device _____ fold(C)
database functions dbm(S) firstkey(S) store(S) fetch(S) perform ___ dbm(S)
OMF to COFF fixobj(CP) convert an object file from __ fixobj(CP)
directories fleece(C) look for files in home _____ fleece(C)
fpgetround(S) fpgetmask(S) IEEE floating point environment control _____ fpgetround(S)
fpgetround(S) fpgetsticky(S) IEEE floating point environment control _____ fpgetround(S)

fpgetround(S) fpsetmask(S) IEEE floating point environment control _____ fpgetround(S)
fpgetround(S) fpsetround(S) IEEE floating point environment control _____ fpgetround(S)
fpgetround(S) fpsetsticky(S) IEEE floating point environment control _____ fpgetround(S)
isnan(S) isnanf(S) isnand(S) test for floating point NaN _____ isnan(S)
ecvt(S) convert floating-point number to string _____ ecvt(S)
modf(S) ldexp(S) manipulate parts of floating-point numbers frexp(S) _____ frexp(S)
functions floor(S) ceil(S) fabs(S) floor, ceiling, and absolute value _____ floor(S)
functions floor(S) fmod(S) floor, ceiling, and absolute value _____ floor(S)
and absolute value functions floor(S) ceil(S) fabs(S) floor, ceiling, floor(S)
absolute value functions floor(S) fmod(S) floor, ceiling, and ____ floor(S)
options(M) floppy disk installation menu _____ options(M)
fcopy(C) copy a floppy diskette _____ fcopy(C)
format(C) format a floppy diskette _____ format(C)
cflow(CP) generate C flow graph _____ cflow(CP)
fclose(S) fflush(S) close or flush a stream _____ fclose(S)
value functions floor(S) fmod(S) floor, ceiling, and absolute ____ floor(S)
fmt(C) simple text formatter _____ fmt(C)
device fold(C) fold long lines for finite width output _ fold(C)
output device fold(C) fold long lines for finite width fold(C)
stream fopen(S) fdopen(S) freopen(S) open a ____ fopen(S)
fork(S) create a new process _____ fork(S)
format(C) format a floppy diskette _____ format(C)
ar(F) archive file format _____ ar(F)
hd(C) display files in hexadecimal format _____ hd(C)
od(C) display files in octal format _____ od(C)
dir(M) format of a directory _____ dir(M)
filesystem(M) format of a system volume _____ filesystem(M)
inode(M) format of an inode _____ inode(M)
sccsfile(F) format of an SCCS file _____ sccsfile(F)
output a.out(F) format of assembler and link editor _____ a.out(F)
menus(M) format of Business Shell menu system ____ menus(M)
core(F) format of core image file _____ core(F)
cpio(F) format of cpio archive _____ cpio(F)
scr_dump(F) format of curses screen image file _____ scr_dump(F)
acct(M) format of per-process accounting file ___ acct(M)
group(M) format of the group file _____ group(M)
utmp(M) wtmp(M) format of utmp and wtmp entries _____ utmp(M)
fspec(F) format specification in text files _____ fspec(F)
syms(F) common object file symbol table format _____ syms(F)
xar(F) archive file format _____ xar(F)
format(C) format a floppy diskette _____ format(C)
intro(F) introduction to file formats _____ intro(F)
scanf(S) fscanf(S) sscanf(S) convert formatted input _____ scanf(S)
vprintf(S) vfprintf(S) vsprintf(S) print formatted output of varargs list _____ vprintf(S)
printf(S) sprintf(S) fprintf(S) print formatted output _____ printf(S)
fmt(C) simple text formatter _____ fmt(C)
convert rational FORTRAN to standard FORTRAN ratfor(CP) _____ ratfor(CP)
ratfor(CP) convert rational FORTRAN to standard FORTRAN _____ ratfor(CP)
environment control fpgetround(S) fpgetmask(S) IEEE floating point _____ fpgetround(S)
point environment control fpgetround(S) fpgetmask(S) IEEE floating fpgetround(S)
floating point environment control fpgetround(S) fpgetsticky(S) IEEE _____ fpgetround(S)
point environment control fpgetround(S) fpsetmask(S) IEEE floating fpgetround(S)
floating point environment control fpgetround(S) fpsetround(S) IEEE _____ fpgetround(S)
floating point environment control fpgetround(S) fpsetsticky(S) IEEE _____ fpgetround(S)
environment control fpgetround(S) fpgetsticky(S) IEEE floating point _____ fpgetround(S)
printf(S) sprintf(S) fprintf(S) print formatted output _____ printf(S)
environment control fpgetround(S) fpsetmask(S) IEEE floating point _____ fpgetround(S)

environment control fpgetround(S) fpsetround(S) IEEE floating point _____ fpgetround(S)
environment control fpgetround(S) fpsetsticky(S) IEEE floating point _____ fpgetround(S)
stream putc(S) putchar(S) putw(S) fputc(S) put character or word on a _____ putc(S)
puts(S) fputs(S) put a string on a stream _____ puts(S)
fwrite(S) fread(S) binary input/output _____ fwrite(S)
tape frec(M) recover files from a back-up ____ frec(M)
df(M) report number of free disk blocks and inodes _____ df(M)
allocator malloc(S) free(S) realloc(S) fast main memory _____ malloc(S)
fopen(S) fdopen(S) freopen(S) open a stream _____ fopen(S)
parts of floating-point numbers frexp(S) modf(S) ldexp(S) manipulate ____ frexp(S)
from(C) list who my mail is from _____ from(C)
gencc(CP) create a front end to the cc command _____ gencc(CP)
input scanf(S) fscanf(S) sscanf(S) convert formatted ___ scanf(S)
list file systems processed by fsck checklist(M) _____ checklist(M)
systems fsck(C) dfsck(C) check and repair file __ fsck(C)
fsdb(M) file system debugger _____ fsdb(M)
file pointer in a stream fseek(S) ftell(S) rewind(S) reposition a fseek(S)
file system fsinfo(M) report information about a ____ fsinfo(M)
files fspec(F) format specification in text ___ fspec(F)
fsplit(CP) split ratfor files _____ fsplit(CP)
fsstat(M) report file system status _____ fsstat(M)
fstab(M) file system table _____ fstab(M)
statfs(S) fstatfs(S) get file system information __ statfs(S)
stat(S) fstat(S) get file status _____ stat(S)
identifier fstyp(M) determine the file system _____ fstyp(M)
pointer in a stream fseek(S) ftell(S) rewind(S) reposition a file ____ fseek(S)
communication package stdipc(S) ftok(S) standard interprocess _____ stdipc(S)
ftw(S) walk a file tree _____ ftw(S)
egrep(C) search file for pattern using full regular expression _____ egrep(C)
more(C) view a file one full screen at a time _____ more(C)
function erf(S) erfc(S) error function and complementary error _____ erf(S)
error function and complementary error function erf(S) erfc(S) _____ erf(S)
gamma(S) log gamma function _____ gamma(S)
hypot(S) Euclidean distance function _____ hypot(S)
manipulate line number entries of a COFF function ldlread(S) ldlitem(S) _____ ldlread(S)
matherr(S) error-handling function _____ matherr(S)
prof(F) profile within a function _____ prof(F)
math(F) math functions and constants _____ math(F)
intro(S) introduce system calls, functions, and libraries _____ intro(S)
bessel(S) j0(S) y0(S) Bessel functions _____ bessel(S)
crypt(S) password and file encryption functions _____ crypt(S)
fetch(S) nextkey(S) perform database functions dbm(S) dbminit(S) _____ dbm(S)
store(S) fetch(S) perform database functions dbm(S) firstkey(S) _____ dbm(S)
log(S) exponential, logarithm, and power functions exp(S) pow(S) _____ exp(S)
exponential, logarithm, and square root functions exp(S) sqrt(S) _____ exp(S)
floor, ceiling, and absolute value functions floor(S) ceil(S) fabs(S) _____ floor(S)
floor, ceiling, and absolute value functions floor(S) fmod(S) _____ floor(S)
sinh(S) cosh(S) tanh(S) hyperbolic functions _____ sinh(S)
trig(S) atan(S) atan2(S) trigonometric functions _____ trig(S)
tan(S) asin(S) acos(S) trigonometric functions trig(S) sin(S) cos(S) _____ trig(S)
or file structure fuser(M) identify processes using a file fuser(M)
fwrite(S) fread(S) binary input/output __ fwrite(S)
files xlist(S) fxlist(S) get name list entries from ____ xlist(S)
gamma(S) log gamma function _____ gamma(S)
gamma(S) log gamma function _____ gamma(S)
command gencc(CP) create a front end to the cc __ gencc(CP)
adb(C) invoke x.out general purpose debugger _____ adb(C)

termio(M) general terminal interface _____ termio(M)
random(C) generate a random number _____ random(C)
mkvers(CP) generate a what string _____ mkvers(CP)
makekey(M) generate an encryption key _____ makekey(M)
abort(S) generate an IOT fault _____ abort(S)
cflow(CP) generate C flow graph _____ cflow(CP)
cxref(CP) generate C program cross-reference _____ cxref(CP)
ctermid(S) generate file name for terminal _____ ctermid(S)
ncheck(M) generate path names from inode numbers __ ncheck(M)
lex(CP) generate programs for lexical analysis __ lex(CP)
drand48(S) erand48(S) generate pseudo-random numbers _____ drand48(S)
/mrand48(S) nrand48(S) lrand48(S) generate pseudo-random numbers _____ drand48(S)
/seed48(S) srand48(S) jrand48(S) generate pseudo-random numbers _____ drand48(S)
rand(S) srand(S) simple random-number generator _____ rand(S)
stream getc(S) getw(S) fgetc(S) getchar(S) get character or word from a _ getc(S)
get(CP) get a version of an SCCS file ___ get(CP)
character or word from a stream getc(S) getw(S) fgetc(S) getchar(S) get _ getc(S)
working directory getcwd(S) get path name of current _____ getcwd(S)
put in a file getdents(S) read directory entries and __ getdents(S)
group IDs getuid(S) getegid(S) get real/effective user or ___ getuid(S)
name getenv(S) return value for environment __ getenv(S)
group IDs getuid(S) geteuid(S) get real/effective user or ___ getuid(S)
group IDs getuid(S) getgid(S) get real/effective user or ____ getuid(S)
setgrent(S) get group file entry getgrent(S) fgetgrent(S) endgrent(S) ____ getgrent(S)
group file entry getgrent(S) getgrnam(S) getgrgid(S) get _ getgrent(S)
getgrent(S) getgrnam(S) getgrgid(S) get group file ___ getgrent(S)
entry getgrent(S) getgrnam(S) getgrgid(S) get group file __ getgrent(S)
getlogin(S) get login name _____ getlogin(S)
getmsg(S) get next message off a stream _ getmsg(S)
getopt(C) parse command options _____ getopt(C)
argument vector getopt(S) get option letter from _____ getopt(S)
getpass(S) read a password _____ getpass(S)
and parent process IDs getpid(S) get process, process group, ___ getpid(S)
setpwent(S) get password file entry getpwent(S) fgetpwent(S) endpwent(S) ____ getpwent(S)
password file entry getpwent(S) getpwnam(S) getpwuid(S) get _ getpwent(S)
file entry getpwent(S) getpwnam(S) getpwuid(S) get password ____ getpwent(S)
getpw(S) get name from UID _____ getpw(S)
getpwent(S) getpwnam(S) getpwuid(S) get password file entry ____ getpwent(S)
input gets(C) get a string from the standard __ gets(C)
stream gets(S) fgets(S) get a string from a ____ gets(S)
speed and terminal settings used by getty gettydefs(M) _____ gettydefs(M)
ct(C) spawn getty to a remote terminal _____ ct(C)
used by getty gettydefs(M) speed and terminal settings gettydefs(M)
getty(M) set terminal mode _____ getty(M)
user or group IDs getuid(S) getegid(S) get real/effective _ getuid(S)
user or group IDs getuid(S) geteuid(S) get real/effective _ getuid(S)
user or group IDs getuid(S) getgid(S) get real/effective __ getuid(S)
access utmp file entry getut(S) getutent(S) utmpname(S) endutent(S) ____ getut(S)
getut(S) setutent(S) getutline(S) access utmp file entry ____ getut(S)
endutent(S) access utmp file entry getut(S) getutent(S) utmpname(S) _____ getut(S)
utmp file entry getut(S) setutent(S) getutline(S) access getut(S)
character or word from a stream getc(S) getw(S) fgetc(S) getchar(S) get _____ getc(S)
login(C) give you system access _____ login(C)
symbols glossary(C) define common UNIX terms and glossary(C)
time to string ctime(S) gmtime(S) localtime(S) convert date and _ ctime(S)
setjmp(S) longjmp(S) non-local goto _____ setjmp(S)
cflow(CP) generate C flow graph _____ cflow(CP)

graph(C) draw a	graph	graph(C)
	graph(C) draw a graph	graph(C)
plot(S)	graphics interface subroutines	plot(S)
	grep(C) search a file for a pattern	grep(C)
getpid(S) get process, process	group, and parent process IDs	getpid(S)
fgetgrent(S) endgrent(S) setgrent(S) get	group file entry getgrent(S)	getgrent(S)
getgrent(S) getgrnam(S) getgrgid(S) get	group file entry	getgrent(S)
group(M) format of the	group file	group(M)
id(C) print user and	group ID and names	id(C)
chown(C) chgrp(C) change owner or	group ID	chown(C)
setpgrp(S) set process	group id	setpgrp(S)
getegid(S) get real/effective user or	group IDs getuid(S)	getuid(S)
geteuid(S) get real/effective user or	group IDs getuid(S)	getuid(S)
getgid(S) get real/effective user or	group IDs getuid(S)	getuid(S)
setuid(S) set user and	group IDs	setuid(S)
newgrp(C) log user into a new	group	newgrp(C)
chown(S) change owner and	group of a file	chown(S)
kill(S) send a signal to a process or a	group of processes	kill(S)
execute command in a new process	group setpgrp(C)	setpgrp(C)
	group(M) format of the group file	group(M)
copy(C) copy	groups of files	copy(C)
make(C) maintain, update, and regenerate	groups of programs	make(C)
tra(C) copy out a file as it	grows	tra(C)
pwck(M)	grpck(M) check password/group file	pwck(M)
ssignal(S)	gsignal(S) software signals	ssignal(S)
haltsys(C) close the file systems and	halt the CPU	haltsys(C)
halt the CPU	haltsys(C) close the file systems and	haltsys(C)
varargs(F)	handles variable argument list	varargs(F)
curses(S) terminal screen	handling and optimization package	curses(S)
nohup(C) run a command immune to	hangups and quits	nohup(C)
add.hd(C) add an additional	hard disk	add.hd(C)
restore.hd(C) restore a	hard disk from tape	restore.hd(C)
layout(M) manage	hard disk partitions	layout(M)
dump.hd(C) dump contents of a	hard disk to tape	dump.hd(C)
upgrade.hd(C) upgrade an additional	hard disk	upgrade.hd(C)
find spelling errors	hashcheck(C)	spell(C)
find spelling errors	hashmake(C)	spell(C)
hsearch(S) hdestroy(S) hcreate(S) manage	hash search tables	hsearch(S)
generate	hashing encryption	crypt(S)
hsearch(S) hdestroy(S)	hcreate(S) manage hash search tables	hsearch(S)
format	hd(C) display files in hexadecimal	hd(C)
search tables hsearch(S)	hdestroy(S) hcreate(S) manage hash	hsearch(S)
object file	hdr(C) display selected parts of an	hdr(C)
stream	head(C) print the first few lines of a	head(C)
scnhdr(F) section	header for a common object file	scnhdr(F)
filehdr(F) file	header for common object files	filehdr(F)
constants limits(F) file	header for implementation-specific	limits(F)
unistd(F) file	header for symbolic constants	unistd(F)
ldfhread(S) read the file	header of a COFF file	ldfhread(S)
read an indexed/named section	header of a COFF file ldshread(S)	ldshread(S)
ldohseek(S) seek to the optional file	header of a common object	ldohseek(S)
ldahread(S) read the archive	header of a member of an archive file	ldahread(S)
help(C) system	help facility	help(C)
	help(C) system help facility	help(C)
hd(C) display files in	hexadecimal format	hd(C)
fleece(C) look for files in	home directories	fleece(C)
printing on LaserJet printer	hplp(C) hplpR(C) filter files for	hplp(C)

LaserJet printer hplp(C) — hplpR(C) filter files for printing on ___ hplp(C)
hash search tables — hsearch(S) hdestroy(S) hcreate(S) manage hsearch(S)
sinh(S) cosh(S) tanh(S) — hyperbolic functions _____ sinh(S)
hypot(S) Euclidean distance function ____ hypot(S)
id(C) print user and group — ID and names _____ id(C)
chown(C) chgrp(C) change owner or group — ID _____ chown(C)
queue, semphore set, shared memory — id ipcrm(C) remove message _____ ipcrm(C)
setpgrp(S) set process group — id _____ setpgrp(S)
whoami(C) print effective current user — id _____ whoami(C)
id(C) print user and group ID and names _ id(C)
fstyp(M) determine the file system — identifier _____ fstyp(M)
shmget(S) get shared memory segment — identifier _____ shmget(S)
reside devnm(C) — identify device name on which files _____ devnm(C)
what(C) — identify files _____ what(C)
structure fuser(M) — identify processes using a file or file _ fuser(M)
process group, and parent process — IDs getpid(S) get process, _____ getpid(S)
get real/effective user or group — IDs getuid(S) getegid(S) _____ getuid(S)
get real/effective user or group — IDs getuid(S) geteuid(S) _____ getuid(S)
get real/effective user or group — IDs getuid(S) getgid(S) _____ getuid(S)
setuid(S) set user and group — IDs _____ setuid(S)
fpgetround(S) fpgetmask(S) — IEEE floating point environment control _ fpgetround(S)
fpgetround(S) fpgetsticky(S) — IEEE floating point environment control _ fpgetround(S)
fpgetround(S) fpsetmask(S) — IEEE floating point environment control _ fpgetround(S)
fpgetround(S) fpsetround(S) — IEEE floating point environment control _ fpgetround(S)
fpgetround(S) fpsetsticky(S) — IEEE floating point environment control _ fpgetround(S)
core(F) format of core — image file _____ core(F)
mem(M) kmem(M) memory — image file _____ mem(M)
scr_dump(F) format of curses screen — image file _____ scr_dump(F)
nohup(C) run a command — immune to hangups and quits _____ nohup(C)
limits(F) file header for — implementation-specific constants _____ limits(F)
finc(M) fast — incremental backup _____ finc(M)
dirent(F) file system — independent directory entry _____ dirent(F)
file ldtbindex(S) compute the — index of a symbol table entry of a COFF _ ldtbindex(S)
file ldtbread(S) read an — indexed symbol table entry of a COFF ____ ldtbread(S)
file ldshread(S) read an — indexed/named section header of a COFF __ ldshread(S)
descriptions — infocmp(M) compare or print terminfo ____ infocmp(M)
fsinfo(M) report — information about a file system _____ fsinfo(M)
finger(C) find — information about users _____ finger(C)
devinfo(C) display device — information _____ devinfo(C)
default(M) default program — information directory _____ default(M)
reloc(F) relocation of — information for a common object file ____ reloc(F)
lpstat(C) print LP status — information _____ lpstat(C)
statfs(S) fstatfs(S) get file system — information _____ statfs(S)
sysconf(C) get system configuration — information _____ sysconf(C)
sysconf(S) get system configuration — information _____ sysconf(S)
sysfs(S) get file system type — information _____ sysfs(S)
uname(C) print the current UNIX — information _____ uname(C)
drive(C) drive — information written during manufacturing drive(C)
executes init — inir(M) clean the file system and _____ inir(M)
clean the file system and executes — init inir(M) _____ inir(M)
inittab(M) script for the — init processes _____ inittab(M)
special login program invoked by — init sulogin(M) _____ sulogin(M)
init(M) process control — initialization _____ init(M)
brc(M) system — initialization procedure _____ brc(M)
popen(S) pclose(S) — initiate pipe to/from a process _____ popen(S)
init(M) process control initialization __ init(M)
inittab(M) script for the init processes inittab(M)

clri(M) clear inode _____ clri(M)
inode(M) format of an inode _____ inode(M)
ncheck(M) generate path names from inode numbers _____ ncheck(M)
inode(M) format of an inode _____ inode(M)

report number of free disk blocks and inodes df(M) _____ df(M)
gets(C) get a string from the standard input _____ gets(C)
line(C) read one line of input _____ line(C)
fscanf(S) sscanf(S) convert formatted input scanf(S) _____ scanf(S)
ungetc(S) push character back into input stream _____ ungetc(S)
fwrite(S) fread(S) binary input/output _____ fwrite(S)
poll(S) STREAMS input/output multiplexing _____ poll(S)
stdio(S) standard buffered input/output package _____ stdio(S)
clearerr(S) feof(S) stream status inquiries ferror(S) fileno(S) _____ ferror(S)
uustat(C) uucp status inquiry and job control _____ uustat(C)
install(M) install commands _____ install(M)
cpset(C) install utilities _____ cpset(C)
options(M) floppy disk installation menu _____ options(M)
install(M) install commands _____ install(M)

abs(S) return integer absolute value _____ abs(S)
a64l(S) l64a(S) convert between long integer and base-64 ASCII string _____ a64l(S)
sputl(S) sgetl(S) access long integer data _____ sputl(S)
atol(S) atoi(S) convert string to integer strtol(S) _____ strtol(S)
l3tol(S) ltol3(S) convert between 3-byte integers and long integers _____ l3tol(S)
convert between 3-byte integers and long integers l3tol(S) ltol3(S) _____ l3tol(S)
plot(S) graphics interface subroutines _____ plot(S)
termio(M) general terminal interface _____ termio(M)
log(M) interface to STREAMS error logging _____ log(M)
spline(C) interpolate smooth curves _____ spline(C)
characters asa(C) interpret asa carriage control _____ asa(C)
sh(C) rsh(C) invoke the shell command interpreter _____ sh(C)
csh(C) shell command interpreter with C-like syntax _____ csh(C)
pipe(S) create an interprocess channel _____ pipe(S)
status ipcs(C) report inter-process communication facilities __ ipcs(C)
stdipc(S) ftok(S) standard interprocess communication package _____ stdipc(S)
nap(S) suspend execution for a short interval _____ nap(S)
sleep(C) suspend execution for an interval _____ sleep(C)
sleep(S) suspend execution for interval _____ sleep(S)
intro(C) introduce commands _____ intro(C)

commands intro(CP) introduce software development intro(CP)
intro(C) introduce commands _____ intro(C)
files intro(M) introduce miscellaneous features and ____ intro(M)
intro(CP) introduce software development commands _ intro(CP)
libraries intro(S) introduce system calls, functions, and __ intro(S)
intro(F) introduction to file formats _____ intro(F)
intro(F) introduction to file formats ___ intro(F)

features and files intro(M) introduce miscellaneous _____ intro(M)
functions, and libraries intro(S) introduce system calls, _____ intro(S)
yacc(CP) invoke a compiler-compiler _____ yacc(CP)
m4(CP) invoke a macro processor _____ m4(CP)
calendar(C) invoke a reminder service _____ calendar(C)
vi(C) invoke a screen-oriented display editor _ vi(C)
ex(C) invoke a text editor _____ ex(C)
bsh(C) invoke the Business shell _____ bsh(C)
cc(CP) invoke the C compiler _____ cc(CP)
ed(C) red(C) invoke the ed text editor _____ ed(C)
edit(C) invoke the edit text editor _____ edit(C)
ld(CP) invoke the link editor _____ ld(CP)

xld(CP) invoke the link editor _____ xld(CP)
masm(CP) invoke the macro assembler _____ masm(CP)
sh(C) rsh(C) invoke the shell command interpreter ____ sh(C)
sed(C) invoke the stream editor _____ sed(C)
adb(C) invoke x.out general purpose debugger ___ adb(C)
sulogin(M) special login program invoked by init _____ sulogin(M)
ioctl(S) control device _____ ioctl(S)
abort(S) generate an IOT fault _____ abort(S)
set, shared memory id ipcrm(C) remove message queue, semphore _ ipcrm(C)
communication facilities status ipcs(C) report inter-process _____ ipcs(C)
classify characters ctype(S) isalpha(S) islower(S) iscntrl(S) _____ ctype(S)
ctype(S) isdigit(S) ispunct(S) isascii(S) classify characters _____ ctype(S)
ttyname(S) isatty(S) find name of a terminal _____ ttyname(S)
ctype(S) isalpha(S) islower(S) iscntrl(S) classify characters _____ ctype(S)
classify characters ctype(S) isdigit(S) ispunct(S) isascii(S) _____ ctype(S)
characters ctype(S) isalpha(S) islower(S) iscntrl(S) classify _____ ctype(S)
isnan(S) isnanf(S) isnand(S) test for floating point NaN ___ isnan(S)
point NaN isnan(S) isnanf(S) isnand(S) test for floating ___ isnan(S)
floating point NaN isnan(S) isnanf(S) isnand(S) test for ___ isnan(S)
characters ctype(S) isdigit(S) ispunct(S) isascii(S) classify _____ ctype(S)
system(S) issue a shell command _____ system(S)
bessel(S) j0(S) y0(S) Bessel functions _____ bessel(S)
uustat(C) uucp status inquiry and job control _____ uustat(C)
join(C) join two relations _____ join(C)
join(C) join two relations _____ join(C)
numbers drand48(S) seed48(S) srand48(S) jrand48(S) generate pseudo-random _____ drand48(S)
ldunix(M) configurable kernel linker _____ ldunix(M)
mkunix(M) make bootable system file with kernel symbol table _____ mkunix(M)
makekey(M) generate an encryption key _____ makekey(M)
killall(C) kill all active processes _____ killall(C)
killall(C) kill all active processes ____ killall(C)
kill(C) terminate a process _____ kill(C)
group of processes kill(S) send a signal to a process or a _ kill(S)
mem(M) kmem(M) memory image file _____ mem(M)
integers and long integers l3tol(S) ltol3(S) convert between 3-byte l3tol(S)
base-64 ASCII string a64l(S) l64a(S) convert between long integer and a64l(S)
labelit(M) copy file system with label checking volcopy(M) _____ volcopy(M)
systems labelit(C) provide labels for file _____ labelit(C)
checking volcopy(M) labelit(M) copy file system with label __ volcopy(M)
labelit(C) provide labels for file systems _____ labelit(C)
awk(C) pattern scanning and processing language _____ awk(C)
bc(C) arbitrary-precision arithmetic language _____ bc(C)
nawk(C) pattern scanning and processing language _____ nawk(C)
cpp(CP) the C Language Preprocessor _____ cpp(CP)
lint(CP) check C language usage and syntax _____ lint(CP)
bdiff(C) compare files too large for diff _____ bdiff(C)
banner(C) print large letters _____ banner(C)
hplpR(C) filter files for printing on LaserJet printer hplp(C) _____ hplp(C)
last(C) print last record of user logins last(C)
at(C) batch(C) execute commands at a later time _____ at(C)
layout(M) manage hard disk partitions ___ layout(M)
ldclose(S) ldaclose(S) close a COFF file _____ ldclose(S)
member of an archive file ldahread(S) read the archive header of a ldahread(S)
reading ldopen(S) ldaopen(S) open a common object file for ldopen(S)
ldclose(S) ldaclose(S) close a COFF file ldclose(S)
ld(CP) invoke the link editor _____ ld(CP)
floating-point numbers frexp(S) modf(S) ldexp(S) manipulate parts of _____ frexp(S)

routines ldfcn(F) common object file access ldfcn(F)
COFF file ldfhread(S) read the file header of a ldfhread(S)
COFF symbol table entry ldgetname(S) retrieve symbol name for ldgetname(S)
entries of a COFF function ldlread(S) ldlitem(S) manipulate line number ldlread(S)
number entries of a COFF function ldlread(S) ldlitem(S) manipulate line ldlread(S)
of a section of a COFF file ldlseek(S) seek to line number entries ldlseek(S)
header of a common object ldohseek(S) seek to the optional file ldohseek(S)
object file for reading ldopen(S) ldaopen(S) open a common ldopen(S)
a section of a COFF file ldrseek(S) seek to relocation entries of ldrseek(S)
section header of a COFF file ldshread(S) read an indexed/named ldshread(S)
symbol table entry of a COFF file ldtbindex(S) compute the index of a ldtbindex(S)
entry of a COFF file ldtbread(S) read an indexed symbol table ldtbread(S)
a COFF file ldtbseek(S) seek to the symbol table of ldtbseek(S)
ldunix(M) configurable kernel linker ldunix(M)

leave(C) remind you when you have to leave leave(C)
leave leave(C) remind you when you have to leave(C)
getopt(S) get option letter from argument vector getopt(S)
banner(C) print large letters banner(C)
analysis lex(CP) generate programs for lexical lex(CP)
lex(CP) generate programs for lexical analysis lex(CP)
lsearch(S) lfind(S) linear search and update lsearch(S)
ar(CP) maintain archives and libraries ar(CP)
chkshlib(CP) tool for comparing shared libraries chkshlib(CP)
introduce system calls, functions, and libraries intro(S) intro(S)
ranlib(CP) convert archives to random libraries ranlib(CP)
xar(CP) maintain archives and libraries xar(CP)
find ordering relation for object library lorder(CP) lorder(CP)
mkshlib(CP) create a shared library mkshlib(CP)
shuttype(S) get and set UPS shutdown limits shuttype(S)
ulimit(S) get and set user limits ulimit(S)
implementation-specific constants limits(F) file header for limits(F)
dial(S) establish an out-going terminal line connection dial(S)
set terminal type, modes, speed, line discipline uugetty(M) uugetty(M)
file linenum(F) line number entries in a common object linenum(F)
ldlread(S) ldlitem(S) manipulate line number entries of a COFF function ldlread(S)
COFF file ldlseek(S) seek to line number entries of a section of a ldlseek(S)
strip(CP) remove symbols and line numbers from COFF file strip(CP)
nl(C) add line numbers to a file nl(C)
line(C) read one line of input line(C)
lpd(M) line printer daemon lpd(M)
cancel(C) send/cancel requests to LP line printer lp(C) lp(C)
turn on/off line printer scheduler lpon(M)
lpdisable(C) enable/disable LP line printers lpenable(C) lpenable(C)
lpinit(M) add new line printers lpinit(M)
lsearch(S) lfind(S) linear search and update lsearch(S)
line(C) read one line of input line(C)
common object file linenum(F) line number entries in a linenum(F)
comm(C) select/reject lines common to two sorted files comm(C)
fold(C) fold long lines for finite width output device fold(C)
uniq(C) report repeated lines in a file uniq(C)
look(C) find lines in a sorted list look(C)
num(C) number lines num(C)
rev(C) reverse lines of a file rev(C)
head(C) print the first few lines of a stream head(C)
ssp(C) remove consecutive blank lines ssp(C)
wc(C) count lines, words, and characters wc(C)
link(M) unlink(M) link and unlink files and directories link(M)

ld(CP) invoke the — link editor — ld(CP)
a.out(F) format of assembler and — link editor output — a.out(F)
xld(CP) invoke the — link editor — xld(CP)
link(S) — link to a file — link(S)
ln(C) make a — link to a file — ln(C)
ldunix(M) configurable kernel — linker — ldunix(M)
and directories — link(M) unlink(M) link and unlink files — link(M)
link(S) link to a file — link(S)
syntax — lint(CP) check C language usage and — lint(CP)
ls(C) — list contents of directories — ls(C)
xlist(S) fxlist(S) get name — list entries from files — xlist(S)
checklist(M) — list file systems processed by fsck — checklist(M)
look(C) find lines in a sorted — list — look(C)
nlist(S) get entries from name — list — nlist(S)
nm(CP) print name — list of common object file — nm(CP)
terminals(M) — list of supported terminals — terminals(M)
varargs(F) handles variable argument — list — varargs(F)
print formatted output of varargs — list vprintf(S) vfprintf(S) vsprintf(S) — vprintf(S)
from(C) — list who my mail is from — from(C)
xnm(CP) print name — list — xnm(CP)
COFF file — list(CP) produce C source listing from — list(CP)
cref(CP) make a cross-reference — listing — cref(CP)
list(CP) produce C source — listing from COFF file — list(CP)
ln(C) make a link to a file — ln(C)
string ctime(S) gmtime(S) — localtime(S) convert date and time to — ctime(S)
program whereis(C) — locate source, binary, or manual for — whereis(C)
end(S) edata(S) etext(S) last — locations in program — end(S)
lock(S) — lock a process in primary memory — lock(S)
plock(S) — lock process, text, or data in memory — plock(S)
lockf(S) record locking on files — lockf(S)
lockf(S) record — locking on files — lockf(S)
read/write — locking(S) lock/unlock a file region for — locking(S)
lock(S) lock a process in primary memory — lock(S)
locking(S) — lock/unlock a file region for read/write — locking(S)
errprint(M) display error — log contents — errprint(M)
gamma(S) — log gamma function — gamma(S)
set maximum number of users allowed to — log in numusers(S) get and — numusers(S)
newgrp(C) — log user into a new group — newgrp(C)
exp(S) pow(S) log(S) exponential, — logarithm, and power functions — exp(S)
exp(S) sqrt(S) exponential, — logarithm, and square root functions — exp(S)
strclean(M) STREAMS error — logger cleanup program — strclean(M)
strerr(M) STREAMS error — logger daemon — strerr(M)
log(M) interface to STREAMS error — logging — log(M)
sizefs(C) determine the size of a — logical disk drive — sizefs(C)
getlogin(S) get — login name — getlogin(S)
logname(C) get — login name — logname(C)
cuserid(S) get character — login name of the user — cuserid(S)
logname(S) return — login name of user — logname(S)
passwd(C) change — login password — passwd(C)
sulogin(M) special — login program invoked by init — sulogin(M)
ttys(M) — login terminals file — ttys(M)
profile(M) set up environment at — login time — profile(M)
login(C) give you system access — login(C)
last(C) print last record of user — logins — last(C)
disable(C) disable — logins on a port — disable(C)
enable(C) enable — logins on a port — enable(C)
logging — log(M) interface to STREAMS error — log(M)

	logname(C) get login name	logname(C)
	logname(S) return login name of user	logname(S)
functions exp(S) pow(S)	log(S) exponential, logarithm, and power	exp(S)
setjmp(S)	longjmp(S) non-local goto	setjmp(S)
fleece(C)	look for files in home directories	fleece(C)
	look(C) find lines in a sorted list	look(C)
object library	lorder(CP) find ordering relation for	lorder(CP)
lp(C) cancel(C) send/cancel requests to	LP line printer	lp(C)
lpenable(C) lpdisable(C) enable/disable	LP line printers	lpenable(C)
lpsched(M) lpshut(M) start/stop the	LP request scheduler	lpsched(M)
lpsched(M) lpmove(M) move	LP requests	lpsched(M)
lpadmin(M) configure the	LP spooling system	lpadmin(M)
lpstat(C) print	LP status information	lpstat(C)
system	lpadmin(M) configure the LP spooling	lpadmin(M)
LP line printer	lp(C) cancel(C) send/cancel requests to	lp(C)
printers lpenable(C)	lpdisable(C) enable/disable LP line	lpenable(C)
	lpd(M) line printer daemon	lpd(M)
LP line printers	lpenable(C) lpdisable(C) enable/disable	lpenable(C)
	lpinit(M) add new line printers	lpinit(M)
lpsched(M)	lpmove(M) move LP requests	lpsched(M)
turn on/off	lpon(M) line printer scheduler	lpon(M)
spooler	lpr(C) route named files to printer	lpr(C)
	lpsched(M) lpmove(M) move LP requests	lpsched(M)
request scheduler	lpsched(M) lpshut(M) start/stop the LP	lpsched(M)
scheduler lpsched(M)	lpshut(M) start/stop the LP request	lpsched(M)
	lpstat(C) print LP status information	lpstat(C)
drand48(S) mrand48(S) nrand48(S)	lrand48(S) generate pseudo-random/	drand48(S)
	ls(C) list contents of directories	ls(C)
update	lsearch(S) lfind(S) linear search and	lsearch(S)
	lseek(S) move read/write file pointer	lseek(S)
and long integers l3tol(S)	ltol3(S) convert between 3-byte integers	l3tol(S)
	m4(CP) invoke a macro processor	m4(CP)
values(F)	machine-dependent values	values(F)
aftp(C) transfer files between Altos	machines	aftp(C)
masm(CP) invoke the	macro assembler	masm(CP)
m4(CP) invoke a	macro processor	m4(CP)
enroll(C) xsend(C) xget(C) secret	mail	enroll(C)
mail(C) system	mail	mail(C)
aliases(M)	mail alias file	aliases(M)
aliashash(M) rebuild data base for	mail alias file	aliashash(M)
	mail(C) system mail	mail(C)
from(C) list who my	mail is from	from(C)
malloc(S)	main memory allocator	malloc(S)
malloc(S) free(S) realloc(S) fast	main memory allocator	malloc(S)
mallinfo(S) mallopt(S) calloc(S) fast	main memory allocator malloc(S)	malloc(S)
ar(CP)	maintain archives and libraries	ar(CP)
xar(CP)	maintain archives and libraries	xar(CP)
of programs make(C)	maintain, update, and regenerate groups	make(C)
groups of programs	make(C) maintain, update, and regenerate	make(C)
	makedevs(M) create special device files	makedevs(M)
	makekey(M) generate an encryption key	makekey(M)
	makettys(M) create tty special files	makettys(M)
main memory allocator malloc(S)	mallinfo(S) mallopt(S) calloc(S) fast	malloc(S)
memory allocator	malloc(S) free(S) realloc(S) fast main	malloc(S)
	malloc(S) main memory allocator	malloc(S)
calloc(S) fast main memory allocator	malloc(S) mallinfo(S) mallopt(S)	malloc(S)
allocator malloc(S) mallinfo(S)	mallopt(S) calloc(S) fast main memory	malloc(S)

tsearch(S) tfind(S) tdelete(S) twalk(S) manage binary search trees _____ tsearch(S)
layout(M) manage hard disk partitions _____ layout(M)
hsearch(S) hdestroy(S) hcreate(S) manage hash search tables _____ hsearch(S)
crontab(C) manage user crontab files _____ crontab(C)
sigrelse(S) sigignore(S) signal management sigset(S) sighold(S) _____ sigset(S)
sigset(S) sigpause(S) signal management _____ sigset(S)
function ldlread(S) ldlitem(S) manipulate line number entries of a COFF ldlread(S)
numbers frexp(S) modf(S) ldexp(S) manipulate parts of floating-point _____ frexp(S)
section mcs(CP) manipulate the object file comment _____ mcs(CP)
whereis(C) locate source, binary, or manual for program _____ whereis(C)
sysaltos(S) manufacturer specific system requests ___ sysaltos(S)
drive information written during manufacturing drive(C) _____ drive(C)
add new bad sectors to the bad sector map badblock(C) _____ badblock(C)
ascii(M) map of the ASCII character set _____ ascii(M)
umask(C) set file-creation mode mask _____ umask(C)
umask(S) set and get file creation mask _____ umask(S)
masm(CP) invoke the macro assembler _____ masm(CP)
master(M) master configuration database _____ master(M)
master(M) master configuration database _ master(M)
regexp(F) regular expression compile and match routines _____ regexp(F)
regexp(S) compile regular expression and match routines _____ regexp(S)
math(F) math functions and constants _____ math(F)
matherr(S) error-handling function _____ matherr(S)
math(F) math functions and constants ____ math(F)
in numusers(S) get and set maximum number of users allowed to log __ numusers(S)
comment section mcs(CP) manipulate the object file _____ mcs(CP)
ldahread(S) read the archive header of a member of an archive file _____ ldahread(S)
memory(S) memccpy(S) memory operations _____ memory(S)
memory(S) memset(S) memcpy(S) memcmp(S) memchr(S) memory operations _____ memory(S)
memory(S) memset(S) memcpy(S) memcmp(S) memchr(S) memory operations ___ memory(S)
operations memory(S) memset(S) memcpy(S) memcmp(S) memchr(S) memory ____ memory(S)
mem(M) kmem(M) memory image file _____ mem(M)
malloc(S) free(S) realloc(S) fast main memory allocator _____ malloc(S)
malloc(S) main memory allocator _____ malloc(S)
mallopt(S) calloc(S) fast main memory allocator malloc(S) mallinfo(S) __ malloc(S)
shmctl(S) shared memory control operations _____ shmctl(S)
message queue, semphore set, shared memory id ipcrm(C) remove _____ ipcrm(C)
mem(M) kmem(M) memory image file _____ mem(M)
lock(S) lock a process in primary memory _____ lock(S)
memory(S) memccpy(S) memory operations _____ memory(S)
memset(S) memcpy(S) memcmp(S) memchr(S) memory operations memory(S) _____ memory(S)
shmop(S) shared memory operations _____ shmop(S)
plock(S) lock process, text, or data in memory _____ plock(S)
shmget(S) get shared memory segment identifier _____ shmget(S)
memory(S) memccpy(S) memory operations __ memory(S)
memchr(S) memory operations memory(S) memset(S) memcpy(S) memcmp(S) _ memory(S)
memory operations memory(S) memset(S) memcpy(S) memcmp(S) memchr(S) _ memory(S)
options(M) floppy disk installation menu _____ options(M)
menus(M) format of Business Shell menu system _____ menus(M)
digest(C) create menu system(s) for the Business Shell ___ digest(C)
system menus(M) format of Business Shell menu __ menus(M)
sort(C) sort and merge files _____ sort(C)
to a terminal mesg(C) allow or disallow messages sent _ mesg(C)
msgctl(S) message control operations _____ msgctl(S)
mkstr(C) create an error message file from C source _____ mkstr(C)
mkstr(CP) create an error message file from C source _____ mkstr(CP)
getmsg(S) get next message off a stream _____ getmsg(S)

putmsg(S) send a message on a stream ____ putmsg(S)
msgop(S) message operations ____ msgop(S)
msgget(S) get message queue ____ msgget(S)
memory id ipcrm(C) remove message queue, semphore set, shared ____ ipcrm(C)
perror(S) system error messages ____ perror(S)
mesg(C) allow or disallow messages sent to a terminal ____ mesg(C)
strace(M) print STREAMS trace messages ____ strace(M)
sys_errlist(S) errno(S) system error messages sys_nerr(S) ____ sys_nerr(S)
clone(M) open any minor device on STREAMS driver ____ clone(M)
intro(M) introduce miscellaneous features and files ____ intro(M)
bootable object file mkboot(M) convert object file to ____ mkboot(M)
mkdir(C) make a directory ____ mkdir(C)
mkdir(S) make a directory ____ mkdir(S)
mkfs(M) construct a file system ____ mkfs(M)
mknod(C) build special files ____ mknod(C)
or ordinary file mknod(S) make a directory, or a special _ mknod(S)
mkshlib(CP) create a shared library ____ mkshlib(CP)
from C source mkstr(C) create an error message file ___ mkstr(C)
from C source mkstr(CP) create an error message file __ mkstr(CP)
mktemp(S) make a unique file name ____ mktemp(S)
driver symbol table mkunix(M) make bootable system file with mkunix(M)
kernel symbol table mkunix(M) make bootable system file with mkunix(M)
mkvers(CP) generate a what string ____ mkvers(CP)
mnttab(M) mounted file system table ____ mnttab(M)
getty(M) set terminal mode ____ getty(M)
umask(C) set file-creation mode mask ____ umask(C)
bring system up multi/single-user mode multiuser(C) singleuser(C) ____ multiuser(C)
chmod(S) change mode of file ____ chmod(S)
setmodem(C) set up tty port for a modem ____ setmodem(C)
uugetty(M) set terminal type, modes, speed, line discipline ____ uugetty(M)
tset(C) set terminal modes ____ tset(C)
setmode(C) printer modes utility ____ setmode(C)
floating-point numbers frexp(S) modf(S) ldexp(S) manipulate parts of ____ frexp(S)
settime(C) change the access and modification dates of files ____ settime(C)
touch(C) update access and modification times of a file ____ touch(C)
utime(S) set file access and modification times ____ utime(S)
monitor(S) prepare execution profile ____ monitor(S)
time more(C) view a file one full screen at a more(C)
mount(S) mount a file system ____ mount(S)
multiple file systems mountall(C) umountall(C) mount/unmount __ mountall(C)
structure mount(C) umount(C) mount/unmount a file _ mount(C)
mnttab(M) mounted file system table ____ mnttab(M)
mount(S) mount a file system ____ mount(S)
mount(C) umount(C) mount/unmount a file structure ____ mount(C)
mountall(C) umountall(C) mount/unmount multiple file systems ____ mountall(C)
lpsched(M) lpmove(M) move LP requests ____ lpsched(M)
lseek(S) move read/write file pointer ____ lseek(S)
mv(C) move (rename) files and directories ____ mv(C)
generate pseudo-random/ drand48(S) mrand48(S) nrand48(S) lrand48(S) ____ drand48(S)
dos(C) access MS-DOS files ____ dos(C)
msgctl(S) message control operations ____ msgctl(S)
msgget(S) get message queue ____ msgget(S)
msgop(S) message operations ____ msgop(S)
mountall(C) umountall(C) mount/unmount multiple file systems ____ mountall(C)
poll(S) STREAMS input/output multiplexing ____ poll(S)
singleuser(C) bring system up multi/single-user mode multiuser(C) ____ multiuser(C)
rc2(M) commands for multi-user environment ____ rc2(M)

up multi/single-user mode — multiuser(C) singleuser(C) bring system _ multiuser(C)
directories — mv(C) move (rename) files and _____ mv(C)
tmpnam(S) tempnam(S) create a — name for a temporary file _____ tmpnam(S)
ldgetname(S) retrieve symbol — name for COFF symbol table entry _____ ldgetname(S)
ctermid(S) generate file — name for terminal _____ ctermid(S)
getpw(S) get — name from UID _____ getpw(S)
getenv(S) return value for environment — name _____ getenv(S)
getlogin(S) get login — name _____ getlogin(S)
xlist(S) fxlist(S) get — name list entries from files _____ xlist(S)
nlist(S) get entries from — name list _____ nlist(S)
nm(CP) print — name list of common object file _____ nm(CP)
xnm(CP) print — name list _____ xnm(CP)
logname(C) get login — name _____ logname(C)
mktemp(S) make a unique file — name _____ mktemp(S)
ttyname(S) isatty(S) find — name of a terminal _____ ttyname(S)
uname(S) get — name of current UNIX system _____ uname(S)
getcwd(S) get path — name of current working directory _____ getcwd(S)
cuserid(S) get character login — name of the user _____ cuserid(S)
logname(S) return login — name of user _____ logname(S)
devnm(C) identify device — name on which files reside _____ devnm(C)
pwd(C) print working directory — name _____ pwd(C)
tty(C) get the current port — name _____ tty(C)
lpr(C) route — named files to printer spooler _____ lpr(C)
term(M) conventional — names for terminals _____ term(M)
ncheck(M) generate path — names from inode numbers _____ ncheck(M)
id(C) print user and group ID and — names _____ id(C)
isnand(S) test for floating point — NaN isnan(S) isnanf(S) _____ isnan(S)
interval — nap(S) suspend execution for a short ____ nap(S)
language — nawk(C) pattern scanning and processing _ nawk(C)
semaphore resource waitsem(S) — nbwaitsem(S) wait and check access to ___ waitsem(S)
numbers — ncheck(M) generate path names from inode ncheck(M)
newgrp(C) log user into a new group _____ newgrp(C)
getmsg(S) get — next message off a stream _____ getmsg(S)
dbm(S) dbminit(S) fetch(S) — nextkey(S) perform database functions ___ dbm(S)
priority — nice(C) run a command at a different ____ nice(C)
nice(S) change priority of a process ____ nice(S)
nl(C) add line numbers to a file _____ nl(C)
nlist(S) get entries from name list _____ nlist(S)
file — nm(CP) print name list of common object _ nm(CP)
and quits — nohup(C) run a command immune to hangups nohup(C)
setjmp(S) longjmp(S) — non-local goto _____ setjmp(S)
false(C) return with a — nonzero exit value _____ false(C)
pseudo-random/ drand48(S) mrand48(S) — nrand48(S) lrand48(S) generate _____ drand48(S)
null(M) — null file _____ null(M)
null(M) null file _____ null(M)
linenum(F) line — number entries in a common object file __ linenum(F)
ldlread(S) ldlitem(S) manipulate line — number entries of a COFF function _____ ldlread(S)
file ldlseek(S) seek to line — number entries of a section of a COFF ___ ldlseek(S)
factor(C) factor a — number _____ factor(C)
num(C) — number lines _____ num(C)
df(M) report — number of free disk blocks and inodes ___ df(M)
numusers(S) get and set maximum — number of users allowed to log in _____ numusers(S)
random(C) generate a random — number _____ random(C)
convert string to double-precision — number strtod(S) atof(S) _____ strtod(S)
ecvt(S) convert floating-point — number to string _____ ecvt(S)
erand48(S) generate pseudo-random — numbers drand48(S) _____ drand48(S)
lrand48(S) generate pseudo-random — numbers /mrand48(S) nrand48(S) _____ drand48(S)

jrand48(S) generate pseudo-random numbers drand48(S) seed48(S) srand48(S) _ drand48(S)
manipulate parts of floating-point numbers frexp(S) modf(S) ldexp(S) _____ frexp(S)
strip(CP) remove symbols and line numbers from COFF file _____ strip(CP)
ncheck(M) generate path names from inode numbers _____ ncheck(M)
nl(C) add line numbers to a file _____ nl(C)
num(C) number lines _____ num(C)
of users allowed to log in numusers(S) get and set maximum number __ numusers(S)
dis(CP) object code disassembler _____ dis(CP)
ldfcn(F) common object file access routines _____ ldfcn(F)
mcs(CP) manipulate the object file comment section _____ mcs(CP)
cprs(CP) compresse a common object file _____ cprs(CP)
dump(CP) dump selected parts of an object file _____ dump(CP)
ldopen(S) ldaopen(S) open a common object file for reading _____ ldopen(S)
fixobj(CP) convert an object file from OMF to COFF _____ fixobj(CP)
hdr(C) display selected parts of an object file _____ hdr(C)
line number entries in a common object file linenum(F) _____ linenum(F)
convert object file to bootable object file mkboot(M) _____ mkboot(M)
nm(CP) print name list of common object file _____ nm(CP)
relocation of information for a common object file reloc(F) _____ reloc(F)
scnhdr(F) section header for a common object file _____ scnhdr(F)
find the printable strings in an object file strings(C) _____ strings(C)
syms(F) common object file symbol table format _____ syms(F)
mkboot(M) convert object file to bootable object file _____ mkboot(M)
conv(CP) convert common object files _____ conv(CP)
filehdr(F) file header for common object files _____ filehdr(F)
size(C) print section sizes of common object files _____ size(C)
to the optional file header of a common object ldohseek(S) seek _____ ldohseek(S)
lorder(CP) find ordering relation for object library _____ lorder(CP)
od(C) display files in octal format _____ od(C)
od(C) display files in octal format _____ od(C)
fixobj(CP) convert an object file from OMF to COFF _____ fixobj(CP)
ldopen(S) ldaopen(S) open a common object file for reading ___ ldopen(S)
opensem(S) open a semaphore _____ opensem(S)
fopen(S) fdopen(S) freopen(S) open a stream _____ fopen(S)
clone(M) open any minor device on STREAMS driver _ clone(M)
dup(S) dup2(S) duplicate an open file descriptor _____ dup(S)
open(S) open for reading or writing _____ open(S)
directory(S) telldir(S) readdir(S) opendir(S) directory operations _____ directory(S)
open(S) open for reading or writing _____ open(S)
opensem(S) open a semaphore _____ opensem(S)
rc0(M) commands to stop the operating system _____ rc0(M)
rewinddir(S) seekdir(S) directory operations directory(S) closedir(S) _____ directory(S)
readdir(S) opendir(S) directory operations directory(S) telldir(S) _____ directory(S)
memory(S) memccpy(S) memory operations _____ memory(S)
memcpy(S) memcmp(S) memchr(S) memory operations memory(S) memset(S) _____ memory(S)
msgctl(S) message control operations _____ msgctl(S)
msgop(S) message operations _____ msgop(S)
semctl(S) semaphore control operations _____ semctl(S)
semop(S) semaphore operations _____ semop(S)
shmctl(S) shared memory control operations _____ shmctl(S)
shmop(S) shared memory operations _____ shmop(S)
strdup(S) strpbrk(S) strcmp(S) string operations string(S) strcat(S) _____ string(S)
strcpy(S) strlen(S) strchr(S) string operations string(S) strncmp(S) _____ string(S)
string(S) strspn(S) strtok(S) string operations _____ string(S)
curses(S) terminal screen handling and optimization package _____ curses(S)
getopt(S) get option letter from argument vector _____ getopt(S)
ldohseek(S) seek to the optional file header of a common object _ ldohseek(S)

fcntl(F) file control	options	fcntl(F)
stty(C) set the	options for a port	stty(C)
xtty(C) set the	options for a port	xtty(C)
getopt(C) parse command	options	getopt(C)
getopts(C) parse command	options	getopts(C)
	options(M) floppy disk installation menu	options(M)
lorder(CP) find	ordering relation for object library	lorder(CP)
make a directory, or a special or	ordinary file mknod(S)	mknod(S)
dial(S) establish an	out-going terminal line connection	dial(S)
format of assembler and link editor	output a.out(F)	a.out(F)
fold(C) fold long lines for finite width	output device	fold(C)
vfprintf(S) vsprintf(S) print formatted	output of varargs list vprintf(S)	vprintf(S)
pr(C) print files on the standard	output	pr(C)
sprintf(S) fprintf(S) print formatted	output printf(S)	printf(S)
sysdef(M)	output system definition	sysdef(M)
chown(S) change	owner and group of a file	chown(S)
chown(C) chgrp(C) change	owner or group ID	chown(C)
quot(C) summarize file system	ownership	quot(C)
screen handling and optimization	package curses(S) terminal	curses(S)
sar(M) system activity report	package	sar(M)
stdio(S) standard buffered input/output	package	stdio(S)
standard interprocess communication	package stdipc(S) ftok(S)	stdipc(S)
expand files	pack(C) pcat(C) unpack(C) compress and	pack(C)
tk(C)	paginator for Tektronix 4014	tk(C)
get process, process group, and	parent process IDs getpid(S)	getpid(S)
getopt(C)	parse command options	getopt(C)
getopts(C)	parse command options	getopts(C)
tail(C) deliver the last	part of a file	tail(C)
layout(M) manage hard disk	partitions	layout(M)
dump(CP) dump selected	parts of an object file	dump(CP)
hdr(C) display selected	parts of an object file	hdr(C)
frexp(S) modf(S) ldexp(S) manipulate	parts of floating-point numbers	frexp(S)
	passwd(C) change login password	passwd(C)
	passwd(M) password file	passwd(M)
crypt(S)	password and file encryption functions	crypt(S)
fgetpwent(S) endpwent(S) setpwent(S) get	password file entry getpwent(S)	getpwent(S)
getpwent(S) getpwnam(S) getpwuid(S) get	password file entry	getpwent(S)
putpwent(S) write	password file entry	putpwent(S)
passwd(M)	password file	passwd(M)
getpas(S) read a	password	getpas(S)
passwd(C) change login	password	passwd(C)
pwck(M) grpck(M) check	password/group file	pwck(M)
getcwd(S) get	path name of current working directory	getcwd(S)
ncheck(M) generate	path names from inode numbers	ncheck(M)
dirname(C) deliver portions of	pathnames basename(C)	basename(C)
grep(C) search a file for a	pattern	grep(C)
awk(C)	pattern scanning and processing language	awk(C)
nawk(C)	pattern scanning and processing language	nawk(C)
egrep(C) search file for	pattern using full regular expression	egrep(C)
	pause(S) suspend process until signal	pause(S)
files pack(C)	pcat(C) unpack(C) compress and expand	pack(C)
process popen(S)	pclose(S) initiate pipe to/from a	popen(S)
	pconfig(C) set port configuration	pconfig(C)
dbm(S) dbminit(S) fetch(S) nextkey(S)	perform database functions	dbm(S)
dbm(S) firstkey(S) store(S) fetch(S)	perform database functions	dbm(S)
check the uucp directories and	permissions file uucheck(M)	uucheck(M)
chmod(C) change	permissions of a file or directory	chmod(C)

acct(M) format of	per-process accounting file	acct(M)
	perror(S) system error messages	perror(S)
	pg(C) file perusal filter	pg(C)
split(C) split a file into	pieces	split(C)
tee(C) create a tee in a	pipe	tee(C)
popen(S) pclose(S) initiate	pipe to/from a process	popen(S)
	pipe(S) create an interprocess channel	pipe(S)
memory	plock(S) lock process, text, or data in	plock(S)
	plot(S) graphics interface subroutines	plot(S)
fpgetround(S) fpgetmask(S) IEEE floating	point environment control	fpgetround(S)
fpgetsticky(S) IEEE floating	point environment control fpgetround(S)	fpgetround(S)
fpgetround(S) fpsetmask(S) IEEE floating	point environment control	fpgetround(S)
fpsetround(S) IEEE floating	point environment control fpgetround(S)	fpgetround(S)
fpsetsticky(S) IEEE floating	point environment control fpgetround(S)	fpgetround(S)
isnanf(S) isnand(S) test for floating	point NaN isnan(S)	isnan(S)
ftell(S) rewind(S) reposition a file	pointer in a stream fseek(S)	fseek(S)
lseek(S) move read/write file	pointer	lseek(S)
multiplexing	poll(S) STREAMS input/output	poll(S)
a process	popen(S) pclose(S) initiate pipe to/from	popen(S)
pconfig(C) set	port configuration	pconfig(C)
disable(C) disable logins on a	port	disable(C)
enable(C) enable logins on a	port	enable(C)
setmodem(C) set up tty	port for a modem	setmodem(C)
tty(C) get the current	port name	tty(C)
stty(C) set the options for a	port	stty(C)
xtty(C) set the options for a	port	xtty(C)
basename(C) dirname(C) deliver	portions of pathnames	basename(C)
log(S) exponential, logarithm, and	power functions exp(S) pow(S)	exp(S)
and power functions exp(S)	pow(S) log(S) exponential, logarithm,	exp(S)
	pr(C) print files on the standard output	pr(C)
dc(C) arbitrary	precision calculator	dc(C)
monitor(S)	prepare execution profile	monitor(S)
cpp(CP) the C Language	Preprocessor	cpp(CP)
unget(CP) undo a	previous get of an SCCS file	unget(CP)
lock(S) lock a process in	primary memory	lock(S)
types(F)	primitive system data types	types(F)
cal(C)	print a calendar	cal(C)
yes(C)	print a string repeatedly	yes(C)
prs(CP)	print an SCCS file	prs(CP)
date(C)	print and set the date	date(C)
sact(CP)	print current SCCS file edit activity	sact(CP)
whoami(C)	print effective current user id	whoami(C)
pr(C)	print files on the standard output	pr(C)
vprintf(S) vfprintf(S) vsprintf(S)	print formatted output of varargs list	vprintf(S)
printf(S) sprintf(S) fprintf(S)	print formatted output	printf(S)
banner(C)	print large letters	banner(C)
last(C)	print last record of user logins	last(C)
lpstat(C)	print LP status information	lpstat(C)
nm(CP)	print name list of common object file	nm(CP)
xnm(CP)	print name list	xnm(CP)
printenv(C)	print out the environment	printenv(C)
accept(C) reject(C) allow/prevent	print requests	accept(C)
pscreen(C) set up terminal to	print screen display	pscreen(C)
files size(C)	print section sizes of common object	size(C)
printers(M)	print spooler configuration file	printers(M)
strace(M)	print STREAMS trace messages	strace(M)
infocmp(M) compare or	print terminfo descriptions	infocmp(M)

uname(C) print the current UNIX information _____ uname(C)
head(C) print the first few lines of a stream ___ head(C)
id(C) print user and group ID and names _____ id(C)
pwd(C) print working directory name _____ pwd(C)
strings(C) find the printable strings in an object file _____ strings(C)
printenv(C) print out the environment ___ printenv(C)
lpd(M) line printer daemon _____ lpd(M)
xpd(M) transparent printer daemon _____ xpd(M)
filter files for printing on LaserJet printer hplp(C) hplpR(C) _____ hplp(C)
send/cancel requests to LP line printer lp(C) cancel(C) _____ lp(C)
setmode(C) printer modes utility _____ setmode(C)
turn on/off line printer scheduler _____ lpon(M)
lpr(C) route named files to printer spooler _____ lpr(C)
lpdisable(C) enable/disable LP line printers lpenable(C) _____ lpenable(C)
lpinit(M) add new line printers _____ lpinit(M)
file printers(M) print spooler configuration _ printers(M)
formatted output printf(S) sprintf(S) fprintf(S) print ___ printf(S)
hplp(C) hplpR(C) filter files for printing on LaserJet printer _____ hplp(C)
nice(C) run a command at a different priority _____ nice(C)
nice(S) change priority of a process _____ nice(S)
brc(M) system initialization procedure _____ brc(M)
acct(S) enable or disable process accounting _____ acct(S)
alarm(S) set a process alarm clock _____ alarm(S)
times(S) get process and child process times _____ times(S)
init(M) process control initialization _____ init(M)
exit(S) terminate process _____ exit(S)
fork(S) create a new process _____ fork(S)
getpid(S) get process, process group, and parent process IDs ___ getpid(S)
setpgrp(S) set process group id _____ setpgrp(S)
setpgrp(C) execute command in a new process group _____ setpgrp(C)
get process, process group, and parent process IDs getpid(S) _____ getpid(S)
lock(S) lock a process in primary memory _____ lock(S)
kill(C) terminate a process _____ kill(C)
nice(S) change priority of a process _____ nice(S)
kill(S) send a signal to a process or a group of processes _____ kill(S)
pclose(S) initiate pipe to/from a process popen(S) _____ popen(S)
process IDs getpid(S) get process, process group, and parent _____ getpid(S)
ps(C) report process status _____ ps(C)
plock(S) lock process, text, or data in memory _____ plock(S)
times(S) get process and child process times _____ times(S)
wait(S) wait for child process to stop or terminate _____ wait(S)
ptrace(S) process trace _____ ptrace(S)
pause(S) suspend process until signal _____ pause(S)
checklist(M) list file systems processed by fsck _____ checklist(M)
inittab(M) script for the init processes _____ inittab(M)
killall(C) kill all active processes _____ killall(C)
send a signal to a process or a group of processes kill(S) _____ kill(S)
fuser(M) identify processes using a file or file structure fuser(M)
wait(C) wait completion of background processes _____ wait(C)
awk(C) pattern scanning and processing language _____ awk(C)
nawk(C) pattern scanning and processing language _____ nawk(C)
m4(CP) invoke a macro processor _____ m4(CP)
list(CP) produce C source listing from COFF file _ list(CP)
prof(CP) display profile data _____ prof(CP)
prof(F) profile within a function _____ prof(F)
prof(CP) display profile data _____ prof(CP)
monitor(S) prepare execution profile _____ monitor(S)

profil(S) execution time — profile — profil(S)
prof(F) — profile within a function — prof(F)
time — profile(M) set up environment at login — profile(M)
profil(S) execution time profile — profil(S)
assert(S) verify — program assertion — assert(S)
boot(M) boot — program — boot(M)
cxref(CP) generate C — program cross-reference — cxref(CP)
ctrace(CP) C — program debugger — ctrace(CP)
edata(S) etext(S) last locations in — program end(S) — end(S)
tapeutil(C) utility — program for a streaming tape drive — tapeutil(C)
uucico(M) file transport — program for uucp system — uucico(M)
default(M) default — program information directory — default(M)
sulogin(M) special login — program invoked by init — sulogin(M)
strclean(M) STREAMS error logger cleanup — program — strclean(M)
ua(C) user administration — program — ua(C)
scheduler for the uucp file transport — program uusched(M) — uusched(M)
locate source, binary, or manual for — program whereis(C) — whereis(C)
cb(CP) beautify C — programs — cb(CP)
lex(CP) generate — programs for lexical analysis — lex(CP)
update, and regenerate groups of — programs make(C) maintain, — make(C)
xref(CP) cross-reference C — programs — xref(CP)
xstr(CP) extract strings from C — programs — xstr(CP)
clock(M) — provide access to the time-of-day chip — clock(M)
labelit(C) — provide labels for file systems — labelit(C)
prs(CP) print an SCCS file — prs(CP)
ps(C) report process status — ps(C)
screen display — pscreen(C) set up terminal to print — pscreen(C)
drand48(S) erand48(S) generate — pseudo-random numbers — drand48(S)
nrand48(S) lrand48(S) generate — pseudo-random numbers /mrand48(S) — drand48(S)
seed48(S) srand48(S) jrand48(S) generate — pseudo-random numbers drand48(S) — drand48(S)
ptrace(S) process trace — ptrace(S)
uuto(C) uupick(C) — public UNIX-to-UNIX system file copy — uuto(C)
adb(C) invoke x.out general — purpose debugger — adb(C)
ungetc(S) — push character back into input stream — ungetc(S)
puts(S) fputs(S) — put a string on a stream — puts(S)
putc(S) putchar(S) putw(S) fputc(S) — put character or word on a stream — putc(S)
getdents(S) read directory entries and — put in a file — getdents(S)
character or word on a stream putc(S) — putchar(S) putw(S) fputc(S) put — putc(S)
character or word on a stream — putc(S) putchar(S) putw(S) fputc(S) put — putc(S)
environment — putenv(S) change or add value to — putenv(S)
putmsg(S) send a message on a stream — putmsg(S)
putpwent(S) write password file entry — putpwent(S)
stream — puts(S) fputs(S) put a string on a — puts(S)
on a stream putc(S) putchar(S) — putw(S) fputc(S) put character or word — putc(S)
file — pwck(M) grpck(M) check password/group — pwck(M)
pwd(C) print working directory name — pwd(C)
qsort(S) quicker sort — qsort(S)
query terminfo database — tput(C)
msgget(S) get message — queue — msgget(S)
ipcrm(C) remove message — queue, semphore set, shared memory id — ipcrm(C)
qsort(S) — quicker sort — qsort(S)
run a command immune to hangups and — quits nohup(C) — nohup(C)
quot(C) summarize file system ownership — quot(C)
ranlib(CP) convert archives to — random libraries — ranlib(CP)
random(C) generate a — random number — random(C)
random(C) generate a random number — random(C)
rand(S) srand(S) simple — random-number generator — rand(S)

generator rand(S) srand(S) simple random-number ___ rand(S)
libraries ranlib(CP) convert archives to random ___ ranlib(CP)
fsplit(CP) split ratfor files _____ fsplit(CP)
standard FORTRAN ratfor(CP) convert rational FORTRAN to __ ratfor(CP)
ratfor(CP) convert rational FORTRAN to standard FORTRAN ____ ratfor(CP)
system rc0(M) commands to stop the operating ___ rc0(M)
environment rc2(M) commands for multi-user _____ rc2(M)
to be read rdchk(S) check to see if there is data __ rdchk(S)
getpas(S) read a password _____ getpas(S)
COFF file ldtbread(S) read an indexed symbol table entry of a _ ldtbread(S)
a COFF file ldshread(S) read an indexed/named section header of _ ldshread(S)
getdents(S) read directory entries and put in a file getdents(S)
read(S) read from file _____ read(S)
line(C) read one line of input _____ line(C)
check to see if there is data to be read rdchk(S) _____ rdchk(S)
an archive file ldahread(S) read the archive header of a member of __ ldahread(S)
ldfhread(S) read the file header of a COFF file _____ ldfhread(S)
operations directory(S) telldir(S) readdir(S) opendir(S) directory _____ directory(S)
ldaopen(S) open a common object file for reading ldopen(S) _____ ldopen(S)
open(S) open for reading or writing _____ open(S)
read(S) read from file _____ read(S)
lseek(S) move read/write file pointer _____ lseek(S)
locking(S) lock/unlock a file region for read/write _____ locking(S)
getuid(S) getegid(S) get real/effective user or group IDs _____ getuid(S)
getuid(S) geteuid(S) get real/effective user or group IDs _____ getuid(S)
getuid(S) getgid(S) get real/effective user or group IDs _____ getuid(S)
malloc(S) free(S) realloc(S) fast main memory allocator ___ malloc(S)
autoreboot(C) automatically reboot the system _____ autoreboot(C)
reboot(C) automatically reboot the system _____ reboot(C)
shutdn(S) reboot(S) shutdown or reboot the system _____ shutdn(S)
system reboot(C) automatically reboot the _____ reboot(C)
shutdn(S) reboot(S) shutdown or reboot the system _ shutdn(S)
signal(S) specify what to do on receipt of signal _____ signal(S)
lockf(S) record locking on files _____ lockf(S)
last(C) print last record of user logins _____ last(C)
script(C) make a record of your terminal session _____ script(C)
frec(M) recover files from a back-up tape _____ frec(M)
system from tape recover(C) restore contents of a file ___ recover(C)
ed(C) red(C) invoke the ed text editor _____ ed(C)
regcmp(CP) compile regular expressions __ regcmp(CP)
regcmp(S) compile a regular expression __ regcmp(S)
make(C) maintain, update, and regenerate groups of programs _____ make(C)
match routines regexp(F) regular expression compile and regexp(F)
match routines regexp(S) compile regular expression and regexp(S)
regex(S) execute a regular expression ___ regex(S)
execseg(S) make a data region executable _____ execseg(S)
locking(S) lock/unlock a file region for read/write _____ locking(S)
regexp(S) compile regular expression and match routines ___ regexp(S)
routines regexp(F) regular expression compile and match ____ regexp(F)
search file for pattern using full regular expression egrep(C) _____ egrep(C)
regcmp(S) compile a regular expression _____ regcmp(S)
regex(S) execute a regular expression _____ regex(S)
regcmp(CP) compile regular expressions _____ regcmp(CP)
accept(C) reject(C) allow/prevent print requests __ accept(C)
lorder(CP) find ordering relation for object library _____ lorder(CP)
join(C) join two relations _____ join(C)
COFF file ldrseek(S) seek to relocation entries of a section of a ____ ldrseek(S)

object file reloc(F) relocation of information for a common __ reloc(F)
common object file reloc(F) relocation of information for a reloc(F)
leave(C) remind you when you have to leave _____ leave(C)
calendar(C) invoke a reminder service _____ calendar(C)
uuxqt(M) execute remote command requests _____ uuxqt(M)
uutry(M) contact remote system with debugging on _____ uutry(M)
ct(C) spawn getty to a remote terminal _____ ct(C)
uux(C) execute command on remote UNIX _____ uux(C)
rmdel(CP) remove a delta from an SCCS file _____ rmdel(CP)
rmdir(S) remove a directory _____ rmdir(S)
ssp(C) remove consecutive blank lines _____ ssp(C)
unlink(S) remove directory entry _____ unlink(S)
rm(C) rmdir(C) remove files or directories _____ rm(C)
shared memory id ipcrm(C) remove message queue, semphore set, ____ ipcrm(C)
COFF file strip(CP) remove symbols and line numbers from ____ strip(CP)
mv(C) move (rename) files and directories _____ mv(C)
fsck(C) dfsck(C) check and repair file systems _____ fsck(C)
uniq(C) report repeated lines in a file _____ uniq(C)
yes(C) print a string repeatedly _____ yes(C)
clock(S) report CPU time used _____ clock(S)
fsstat(M) report file system status _____ fsstat(M)
fsinfo(M) report information about a file system __ fsinfo(M)
facilities status ipcs(C) report inter-process communication _____ ipcs(C)
inodes df(M) report number of free disk blocks and ___ df(M)
sar(C) system activity report package _____ sar(C)
sar(M) system activity report package _____ sar(M)
ps(C) report process status _____ ps(C)
uniq(C) report repeated lines in a file _____ uniq(C)
fseek(S) ftell(S) rewind(S) reposition a file pointer in a stream ___ fseek(S)
lpsched(M) lpshut(M) start/stop the LP request scheduler _____ lpsched(M)
accept(C) reject(C) allow/prevent print requests _____ accept(C)
lpsched(M) lpmove(M) move LP requests _____ lpsched(M)
sysaltos(S) manufacturer specific system requests _____ sysaltos(S)
lp(C) cancel(C) send/cancel requests to LP line printer _____ lp(C)
uuxqt(M) execute remote command requests _____ uuxqt(M)
reset(C) reset the teletype bit _____ reset(C)
reset(C) reset the teletype bit _____ reset(C)
identify device name on which files reside devnm(C) _____ devnm(C)
wait and check access to semaphore resource waitsem(S) nbwaitsem(S) _____ waitsem(S)
restore.hd(C) restore a hard disk from tape _____ restore.hd(C)
tape recover(C) restore contents of a file system from __ recover(C)
tape restore.hd(C) restore a hard disk from __ restore.hd(C)
table entry ldgetname(S) retrieve symbol name for COFF symbol ____ ldgetname(S)
stat(F) return data by stat system call _____ stat(F)
abs(S) return integer absolute value _____ abs(S)
logname(S) return login name of user _____ logname(S)
getenv(S) return value for environment name _____ getenv(S)
false(C) return with a nonzero exit value _____ false(C)
true(C) return with a zero exit value _____ true(C)
rev(C) reverse lines of a file _____ rev(C)
rev(C) reverse lines of a file _____ rev(C)
operations directory(S) closedir(S) rewinddir(S) seekdir(S) directory _____ directory(S)
stream fseek(S) ftell(S) rewind(S) reposition a file pointer in a fseek(S)
creat(S) create a new file or rewrite an existing one _____ creat(S)
directories rm(C) rmdir(C) remove files or _____ rm(C)
uucp link rmail(C) receives mail from _____ rmail(C)
file rmdel(CP) remove a delta from an SCCS ___ rmdel(CP)

rm(C)	rmdir(C) remove files or directories	rm(C)
	rmdir(S) remove a directory	rmdir(S)
chroot(S) change	root directory	chroot(S)
chroot(C) change	root directory for command	chroot(C)
exponential, logarithm, and square	root functions exp(S) sqrt(S)	exp(S)
lpr(C)	route named files to printer spooler	lpr(C)
ldfcn(F) common object file access	routines	ldfcn(F)
regular expression compile and match	routines regexp(F)	regexp(F)
compile regular expression and match	routines regexp(S)	regexp(S)
interpreter sh(C)	rsh(C) invoke the shell command	sh(C)
nice(C)	run a command at a different priority	nice(C)
quits nohup(C)	run a command immune to hangups and	nohup(C)
activity	sact(CP) print current SCCS file edit	sact(CP)
system activity	sadcon(M) data collector	sadcon(M)
	sar(C) system activity report package	sar(C)
	sar(M) system activity report package	sar(M)
archive(C)	save a file system to a streaming tape	archive(C)
allocation brk(S)	sbrk(S) change data segment space	brk(S)
bfs(C)	scan big files	bfs(C)
formatted input	scanf(S) fscanf(S) sscanf(S) convert	scanf(S)
awk(C) pattern	scanning and processing language	awk(C)
nawk(C) pattern	scanning and processing language	nawk(C)
cdc(CP) change the delta commentary of	SCCS delta	cdc(CP)
comb(CP) combine	SCCS deltas	comb(CP)
delta(CP) make a change to an	SCCS file	delta(CP)
sact(CP) print current	SCCS file edit activity	sact(CP)
get(CP) get a version of an	SCCS file	get(CP)
prs(CP) print an	SCCS file	prs(CP)
rmdel(CP) remove a delta from an	SCCS file	rmdel(CP)
sccsdiff(CP) compare two versions of an	SCCS file	sccsdiff(CP)
sccsfile(F) format of an	SCCS file	sccsfile(F)
unget(CP) undo a previous get of an	SCCS file	unget(CP)
val(CP) validate an	SCCS file	val(CP)
admin(CP) create and administer	SCCS files	admin(CP)
SCCS file	sccsdiff(CP) compare two versions of an	sccsdiff(CP)
	sccsfile(F) format of an SCCS file	sccsfile(F)
turn on/off	scheduler for line printer	lpon(M)
ckbupscd(M) check file system backup	schedule	ckbupscd(M)
turn on/off	scheduler for line printer	lpon(M)
program uusched(M)	scheduler for the uucp file transport	uusched(M)
lpshut(M) start/stop the LP request	scheduler lpsched(M)	lpsched(M)
object file	scnhdr(F) section header for a common	scnhdr(F)
image file	scr_dump(F) format of curses screen	scr_dump(F)
more(C) view a file one full	screen at a time	more(C)
clear(C) clear terminal	screen	clear(C)
pscreen(C) set up terminal to print	screen display	pscreen(C)
curses(S) terminal	screen handling and optimization package	curses(S)
scr_dump(F) format of curses	screen image file	scr_dump(F)
vi(C) invoke a	screen-oriented display editor	vi(C)
inittab(M)	script for the init processes	inittab(M)
session	script(C) make a record of your terminal	script(C)
	sdb(C) symbolic debugger	sdb(C)
to a shared data segment	sdenter(S) sdleave(S) synchronize access	sdenter(S)
data segment sdget(S)	sdfree(S) attach and detach a shared	sdget(S)
shared data segment	sdget(S) sdfree(S) attach and detach a	sdget(S)
data access	sdgetv(S) sdwaitv(S) synchronize shared	sdgetv(S)
	sdiff(C) compare files side-by-side	sdiff(C)

shared data segment sdenter(S) sdleave(S) synchronize access to a _____ sdenter(S)
access sdgetv(S) sdwaitv(S) synchronize shared data _____ sdgetv(S)
fgrep(C) search a file for a character string ____ fgrep(C)
grep(C) search a file for a pattern _____ grep(C)
lsearch(S) lfind(S) linear search and update _____ lsearch(S)
regular expression egrep(C) search file for pattern using full _____ egrep(C)
bsearch(S) binary search of a sorted table _____ bsearch(S)
hdestroy(S) hcreate(S) manage hash search tables hsearch(S) _____ hsearch(S)
tdelete(S) twalk(S) manage binary search trees tsearch(S) tfind(S) _____ tsearch(S)
enroll(C) xsend(C) xget(C) secret mail _____ enroll(C)
scnhdr(F) section header for a common object file _ scnhdr(F)
ldshread(S) read an indexed/named section header of a COFF file _____ ldshread(S)
manipulate the object file comment section mcs(CP) _____ mcs(CP)
seek to line number entries of a section of a COFF file ldlseek(S) _____ ldlseek(S)
seek to relocation entries of a section of a COFF file ldrseek(S) _____ ldrseek(S)
size(C) print section sizes of common object files ____ size(C)
add new bad sectors to the bad sector map badblock(C) _____ badblock(C)
badblock(C) add new bad sectors to the bad sector map _____ badblock(C)
sed(C) invoke the stream editor _____ sed(C)
see(C) display a file _____ see(C)
pseudo-random numbers drand48(S) seed48(S) srand48(S) jrand48(S) generate drand48(S)
of a COFF file ldlseek(S) seek to line number entries of a section ldlseek(S)
of a COFF file ldrseek(S) seek to relocation entries of a section _ ldrseek(S)
common object ldohseek(S) seek to the optional file header of a ___ ldohseek(S)
ldtbseek(S) seek to the symbol table of a COFF file _ ldtbseek(S)
directory(S) closedir(S) rewinddir(S) seekdir(S) directory operations _____ directory(S)
shmget(S) get shared memory segment identifier _____ shmget(S)
synchronize access to a shared data segment sdenter(S) sdleave(S) _____ sdenter(S)
attach and detach a shared data segment sdget(S) sdfree(S) _____ sdget(S)
brk(S) sbrk(S) change data segment space allocation _____ brk(S)
dump(CP) dump selected parts of an object file _____ dump(CP)
hdr(C) display selected parts of an object file _____ hdr(C)
files comm(C) select/reject lines common to two sorted comm(C)
semctl(S) semaphore control operations _____ semctl(S)
creatsem(S) create a binary semaphore _____ creatsem(S)
opensem(S) open a semaphore _____ opensem(S)
semop(S) semaphore operations _____ semop(S)
nbwaitsem(S) wait and check access to semaphore resource waitsem(S) _____ waitsem(S)
semget(S) get set of semaphores _____ semget(S)
semctl(S) semaphore control operations __ semctl(S)
semget(S) get set of semaphores _____ semget(S)
semop(S) semaphore operations _____ semop(S)
ipcrm(C) remove message queue, semphore set, shared memory id _____ ipcrm(C)
putmsg(S) send a message on a stream _____ putmsg(S)
processes kill(S) send a signal to a process or a group of kill(S)
lp(C) cancel(C) send/cancel requests to LP line printer _ lp(C)
mesg(C) allow or disallow messages sent to a terminal _____ mesg(C)
calendar(C) invoke a reminder service _____ calendar(C)
script(C) make a record of your terminal session _____ script(C)
alarm(S) set a process alarm clock _____ alarm(S)
umask(S) set and get file creation mask _____ umask(S)
ascii(M) map of the ASCII character set _____ ascii(M)
timezone(M) set default system time zone _____ timezone(M)
env(C) set environment for command execution ___ env(C)
utime(S) set file access and modification times __ utime(S)
umask(C) set file-creation mode mask _____ umask(C)
log in numusers(S) get and set maximum number of users allowed to __ numusers(S)

semget(S) get | set of semaphores | semget(S)
pconfig(C) | set port configuration | pconfig(C)
setpgrp(S) | set process group id | setpgrp(S)
ipcrm(C) remove message queue, semphore | set, shared memory id | ipcrm(C)
tabs(C) | set tabs on a terminal | tabs(C)
getty(M) | set terminal mode | getty(M)
tset(C) | set terminal modes | tset(C)
discipline uugetty(M) | set terminal type, modes, speed, line | uugetty(M)
date(C) print and | set the date | date(C)
stty(C) | set the options for a port | stty(C)
xtty(C) | set the options for a port | xtty(C)
asktime(C) | set the system time of day | asktime(C)
stime(S) | set time | stime(S)
profile(M) | set up environment at login time | profile(M)
pscreen(C) | set up terminal to print screen display | pscreen(C)
setmodem(C) | set up tty port for a modem | setmodem(C)
shuttype(S) get and | set UPS shutdown limits | shuttype(S)
setuid(S) | set user and group IDs | setuid(S)
ulimit(S) get and | set user limits | ulimit(S)
a stream | setbuf(S) setvbuf(S) assign buffering to | setbuf(S)
getgrent(S) fgetgrent(S) endgrent(S) | setgrent(S) get group file entry | getgrent(S)
| setjmp(S) longjmp(S) non-local goto | setjmp(S)
| setmnt(C) establish /etc/mnttab table | setmnt(C)
| setmode(C) printer modes utility | setmode(C)
| setmodem(C) set up tty port for a modem | setmodem(C)
process group | setpgrp(C) execute command in a new | setpgrp(C)
| setpgrp(S) set process group id | setpgrp(S)
getpwent(S) fgetpwent(S) endpwent(S) | setpwent(S) get password file entry | getpwent(S)
modification dates of files | settime(C) change the access and | settime(C)
gettydefs(M) speed and terminal | settings used by getty | gettydefs(M)
| setuid(S) set user and group IDs | setuid(S)
file entry getut(S) | setutent(S) getutline(S) access utmp | getut(S)
setbuf(S) | setvbuf(S) assign buffering to a stream | setbuf(S)
sputl(S) | sgetl(S) access long integer data | sputl(S)
sdgetv(S) sdwaitv(S) synchronize | shared data access | sdgetv(S)
sdleave(S) synchronize access to a | shared data segment sdenter(S) | sdenter(S)
sdget(S) sdfree(S) attach and detach a | shared data segment | sdget(S)
chkshlib(CP) tool for comparing | shared libraries | chkshlib(CP)
mkshlib(CP) create a | shared library | mkshlib(CP)
shmctl(S) | shared memory control operations | shmctl(S)
remove message queue, semphore set, | shared memory id ipcrm(C) | ipcrm(C)
shmop(S) | shared memory operations | shmop(S)
shmget(S) get | shared memory segment identifier | shmget(S)
interpreter | sh(C) rsh(C) invoke the shell command | sh(C)
bsh(C) invoke the Business | shell | bsh(C)
sh(C) rsh(C) invoke the | shell command interpreter | sh(C)
syntax csh(C) | shell command interpreter with C-like | csh(C)
system(S) issue a | shell command | system(S)
create menu system(s) for the Business | Shell digest(C) | digest(C)
menus(M) format of Business | Shell menu system | menus(M)
| shl(C) shell layers | shl(C)
operations | shmctl(S) shared memory control | shmctl(S)
identifier | shmget(S) get shared memory segment | shmget(S)
| shmop(S) shared memory operations | shmop(S)
nap(S) suspend execution for a | short interval | nap(S)
the system | shutdn(S) reboot(S) shutdown or reboot | shutdn(S)
shutype(M) UPS | shutdown configuration utility | shutype(M)

shuttype(S) get and set UPS — shutdown limits _____ shuttype(S)
shutdn(S) reboot(S) — shutdown or reboot the system _____ shutdn(S)
bring system to single-user or — shutdown shutdown(M) _____ shutdown(M)
or shutdown — shutdown(M) bring system to single-user _ shutdown(M)
limits — shuttype(S) get and set UPS shutdown ____ shuttype(S)
utility — shutype(M) UPS shutdown configuration ___ shutype(M)
sdiff(C) compare files — side-by-side _____ sdiff(C)
signal management sigset(S) — sighold(S) sigrelse(S) sigignore(S) _____ sigset(S)
sigset(S) sighold(S) sigrelse(S) — sigignore(S) signal management _____ sigset(S)
sighold(S) sigrelse(S) sigignore(S) — signal management sigset(S) _____ sigset(S)
sigset(S) sigpause(S) — signal management _____ sigset(S)
pause(S) suspend process until — signal _____ pause(S)
specify what to do on receipt of — signal signal(S) _____ signal(S)
processes kill(S) send a — signal to a process or a group of _____ kill(S)
of signal — signal(S) specify what to do on receipt _ signal(S)
ssignal(S) gsignal(S) software — signals _____ ssignal(S)
sigset(S) — sigpause(S) signal management _____ sigset(S)
management sigset(S) sighold(S) — sigrelse(S) sigignore(S) signal _____ sigset(S)
sigignore(S) signal management — sigset(S) sighold(S) sigrelse(S) _____ sigset(S)
— sigset(S) sigpause(S) signal management _ sigset(S)
rand(S) srand(S) — simple random-number generator _____ rand(S)
fmt(C) — simple text formatter _____ fmt(C)
shutdown(M) bring system to — single-user or shutdown _____ shutdown(M)
multi/single-user mode multiuser(C) — singleuser(C) bring system up _____ multiuser(C)
functions — sinh(S) cosh(S) tanh(S) hyperbolic _____ sinh(S)
trigonometric functions trig(S) — sin(S) cos(S) tan(S) asin(S) acos(S) ____ trig(S)
chsize(S) change the file — size _____ chsize(S)
sizefs(C) determine the — size of a logical disk drive _____ sizefs(C)
object files — size(C) print section sizes of common ___ size(C)
logical disk drive — sizefs(C) determine the size of a _____ sizefs(C)
size(C) print section — sizes of common object files _____ size(C)
interval — sleep(C) suspend execution for an _____ sleep(C)
— sleep(S) suspend execution for interval _ sleep(S)
user ttyslot(S) find the — slot in the utmp file of the current ____ ttyslot(S)
spline(C) interpolate — smooth curves _____ spline(C)
intro(CP) introduce — software development commands _____ intro(CP)
ssignal(S) gsignal(S) — software signals _____ ssignal(S)
tsort(C) — sort a file topologically _____ tsort(C)
sort(C) — sort and merge files _____ sort(C)
qsort(S) quicker — sort _____ qsort(S)
— sort(C) sort and merge files _____ sort(C)
select/reject lines common to two — sorted files comm(C) _____ comm(C)
look(C) find lines in a — sorted list _____ look(C)
bsearch(S) binary search of a — sorted table _____ bsearch(S)
whereis(C) locate — source, binary, or manual for program ___ whereis(C)
list(CP) produce C — source listing from COFF file _____ list(CP)
create an error message file from C — source mkstr(C) _____ mkstr(C)
create an error message file from C — source mkstr(CP) _____ mkstr(CP)
tic(C) compile terminfo — source _____ tic(C)
brk(S) sbrk(S) change data segment — space allocation _____ brk(S)
ct(C) — spawn getty to a remote terminal _____ ct(C)
makedevs(M) create — special device files _____ makedevs(M)
makettys(M) create tty — special files _____ makettys(M)
mknod(C) build — special files _____ mknod(C)
sulogin(M) — special login program invoked by init ___ sulogin(M)
mknod(S) make a directory, or a — special or ordinary file _____ mknod(S)
sysaltos(S) manufacturer — specific system requests _____ sysaltos(S)

fspec(F) format specification in text files _____ fspec(F)
cron(C) execute commands at specified times _____ cron(C)
signal(S) specify what to do on receipt of signal _ signal(S)
getty gettydefs(M) speed and terminal settings used by _____ gettydefs(M)
uugetty(M) set terminal type, modes, speed, line discipline _____ uugetty(M)
find spelling errors spell(C) _____ spell(C)
spline(C) interpolate smooth curves _____ spline(C)
split(C) split a file into pieces _____ split(C)
csplit(C) split files according to context _____ csplit(C)
fsplit(CP) split ratfor files _____ fsplit(CP)
split(C) split a file into pieces _____ split(C)
uucleanup(M) uucp spool directory cleanup _____ uucleanup(M)
printers(M) print spooler configuration file _____ printers(M)
lpr(C) route named files to printer spooler _____ lpr(C)
lpadmin(M) configure the LP spooling system _____ lpadmin(M)
output printf(S) sprintf(S) fprintf(S) print formatted ___ printf(S)
data sputl(S) sgetl(S) access long integer ___ sputl(S)
square root functions exp(S) sqrt(S) exponential, logarithm, and _____ exp(S)
sqrt(S) exponential, logarithm, and square root functions exp(S) _____ exp(S)
pseudo-random/ drand48(S) seed48(S) srand48(S) jrand48(S) generate _____ drand48(S)
rand(S) srand(S) simple random-number generator _ rand(S)
scanf(S) fscanf(S) sscanf(S) convert formatted input _____ scanf(S)
ssignal(S) gsignal(S) software signals __ ssignal(S)
ssp(C) remove consecutive blank lines ___ ssp(C)
stdio(S) standard buffered input/output package __ stdio(S)
ratfor(CP) convert rational FORTRAN to standard FORTRAN _____ ratfor(CP)
gets(C) get a string from the standard input _____ gets(C)
package stdipc(S) ftok(S) standard interprocess communication _____ stdipc(S)
pr(C) print files on the standard output _____ pr(C)
lpsched(M) lpshut(M) start/stop the LP request scheduler _____ lpsched(M)
stat(F) return data by stat system call _____ stat(F)
stat(F) return data by stat system call _ stat(F)
information statfs(S) fstatfs(S) get file system ____ statfs(S)
ustat(S) get file system statistics _____ ustat(S)
stat(S) fstat(S) get file status _____ stat(S)
fsstat(M) report file system status _____ fsstat(M)
lpstat(C) print LP status information _____ lpstat(C)
fileno(S) clearerr(S) feof(S) stream status inquiries ferror(S) _____ ferror(S)
uustat(C) uucp status inquiry and job control _____ uustat(C)
inter-process communication facilities status ipcs(C) report _____ ipcs(C)
ps(C) report process status _____ ps(C)
stat(S) fstat(S) get file status _____ stat(S)
package stdio(S) standard buffered input/output _ stdio(S)
communication package stdipc(S) ftok(S) standard interprocess _ stdipc(S)
stime(S) set time _____ stime(S)
wait(S) wait for child process to stop or terminate _____ wait(S)
rc0(M) commands to stop the operating system _____ rc0(M)
functions dbm(S) firstkey(S) store(S) fetch(S) perform database _____ dbm(S)
strace(M) print STREAMS trace messages __ strace(M)
string operations string(S) strcat(S) strdup(S) strpbrk(S) strcmp(S) string(S)
string(S) strncmp(S) strcpy(S) strlen(S) strchr(S) string operations _____ string(S)
program strclean(M) STREAMS error logger cleanup strclean(M)
string(S) strcat(S) strdup(S) strpbrk(S) strcmp(S) string operations _____ string(S)
operations string(S) strncmp(S) strcpy(S) strlen(S) strchr(S) string ____ string(S)
operations string(S) strcat(S) strdup(S) strpbrk(S) strcmp(S) string ___ string(S)
sed(C) invoke the stream editor _____ sed(C)
fclose(S) fflush(S) close or flush a stream _____ fclose(S)

fopen(S) fdopen(S) freopen(S) open a — stream _____ — fopen(S)
rewind(S) reposition a file pointer in a — stream fseek(S) ftell(S) _____ — fseek(S)
getchar(S) get character or word from a — stream getc(S) getw(S) fgetc(S) _____ — getc(S)
getmsg(S) get next message off a — stream _____ — getmsg(S)
gets(S) fgets(S) get a string from a — stream _____ — gets(S)
head(C) print the first few lines of a — stream _____ — head(C)
fputc(S) put character or word on a — stream putc(S) putchar(S) putw(S) _____ — putc(S)
putmsg(S) send a message on a — stream _____ — putmsg(S)
puts(S) fputs(S) put a string on a — stream _____ — puts(S)
setvbuf(S) assign buffering to a — stream setbuf(S) _____ — setbuf(S)
ferror(S) fileno(S) clearerr(S) feof(S) — stream status inquiries _____ — ferror(S)
ungetc(S) push character back into input — stream _____ — ungetc(S)
archive(C) save a file system to a — streaming tape _____ — archive(C)
tapeutil(C) utility program for a — streaming tape drive _____ — tapeutil(C)
clone(M) open any minor device on — STREAMS driver _____ — clone(M)
strclean(M) — STREAMS error logger cleanup program ____ — strclean(M)
strerr(M) — STREAMS error logger daemon _____ — strerr(M)
log(M) interface to — STREAMS error logging _____ — log(M)
poll(S) — STREAMS input/output multiplexing _____ — poll(S)
strace(M) print — STREAMS trace messages _____ — strace(M)
strerr(M) STREAMS error logger daemon ___ — strerr(M)
between long integer and base-64 ASCII — string a64l(S) l64a(S) convert _____ — a64l(S)
localtime(S) convert date and time to — string ctime(S) gmtime(S) _____ — ctime(S)
cftime(S) convert date and time to — string ctime(S) tzset(S) asctime(S) _____ — ctime(S)
ecvt(S) convert floating-point number to — string _____ — ecvt(S)
fgrep(C) search a file for a character — string _____ — fgrep(C)
gets(S) fgets(S) get a — string from a stream _____ — gets(S)
gets(C) get a — string from the standard input _____ — gets(C)
mkvers(CP) generate a what — string _____ — mkvers(CP)
puts(S) fputs(S) put a — string on a stream _____ — puts(S)
strcat(S) strdup(S) strpbrk(S) strcmp(S) — string operations string(S) _____ — string(S)
strncmp(S) strcpy(S) strlen(S) strchr(S) — string operations string(S) _____ — string(S)
string(S) strspn(S) strtok(S) — string operations _____ — string(S)
yes(C) print a — string repeatedly _____ — yes(C)
strtod(S) atof(S) convert — string to double-precision number _____ — strtod(S)
strtol(S) atol(S) atoi(S) convert — string to integer _____ — strtol(S)
xstr(CP) extract — strings from C programs _____ — xstr(CP)
strings(C) find the printable — strings in an object file _____ — strings(C)
strcmp(S) string operations — string(S) strcat(S) strdup(S) strpbrk(S) — string(S)
strchr(S) string operations — string(S) strncmp(S) strcpy(S) strlen(S) — string(S)
operations — string(S) strspn(S) strtok(S) string ____ — string(S)
an object file — strings(C) find the printable strings in — strings(C)
numbers from COFF file — strip(CP) remove symbols and line _____ — strip(CP)
string(S) strncmp(S) strcpy(S) — strlen(S) strchr(S) string operations ___ — string(S)
string operations string(S) — strncmp(S) strcpy(S) strlen(S) strchr(S) — string(S)
string(S) strcat(S) strdup(S) — strpbrk(S) strcmp(S) string operations __ — string(S)
string(S) — strspn(S) strtok(S) string operations ___ — string(S)
double-precision number — strtod(S) atof(S) convert string to ____ — strtod(S)
string(S) strspn(S) — strtok(S) string operations _____ ____ — string(S)
to integer — strtol(S) atol(S) atoi(S) convert string — strtol(S)
identify processes using a file or file — structure fuser(M) _____ — fuser(M)
mount(C) umount(C) mount/unmount a file — structure _____ — mount(C)
stty(C) set the options for a port _____ — stty(C)
plot(S) graphics interface — subroutines _____ — plot(S)
another user — su(C) make the user a super-user or _____ — su(C)
by init — sulogin(M) special login program invoked — sulogin(M)
blocks in a file — sum(C) calculate checksum and count _____ — sum(C)

du(C)	summarize disk usage	du(C)
quot(C)	summarize file system ownership	quot(C)
sync(S) update	super block	sync(S)
sync(C) update the	super-block	sync(C)
su(C) make the user a	super-user or another user	su(C)
terminals(M) list of	supported terminals	terminals(M)
nap(S)	suspend execution for a short interval	nap(S)
sleep(C)	suspend execution for an interval	sleep(C)
sleep(S)	suspend execution for interval	sleep(S)
pause(S)	suspend process until signal	pause(S)
	swab(S) swap bytes	swab(S)
swab(S)	swap bytes	swab(S)
swap(C) change	swap device configuration	swap(C)
	swap(C) change swap device configuration	swap(C)
ldgetname(S) retrieve	symbol name for COFF symbol table entry	ldgetname(S)
retrieve symbol name for COFF	symbol table entry ldgetname(S)	ldgetname(S)
ldtbindex(S) compute the index of a	symbol table entry of a COFF file	ldtbindex(S)
ldtbread(S) read an indexed	symbol table entry of a COFF file	ldtbread(S)
syms(F) common object file	symbol table format	syms(F)
make bootable system file with driver	symbol table mkunix(M)	mkunix(M)
make bootable system file with kernel	symbol table mkunix(M)	mkunix(M)
ldtbseek(S) seek to the	symbol table of a COFF file	ldtbseek(S)
unistd(F) file header for	symbolic constants	unistd(F)
sdb(C)	symbolic debugger	sdb(C)
strip(CP) remove	symbols and line numbers from COFF file	strip(CP)
glossary(C) define common UNIX terms and	symbols	glossary(C)
format	syms(F) common object file symbol table	syms(F)
	sync(C) update the super-block	sync(C)
segment sdenter(S) sdleave(S)	synchronize access to a shared data	sdenter(S)
sdgetv(S) sdwaitv(S)	synchronize shared data access	sdgetv(S)
	sync(S) update super block	sync(S)
shell command interpreter with C-like	syntax csh(C)	csh(C)
lint(CP) check C language usage and	syntax	lint(CP)
requests	sysaltos(S) manufacturer specific system	sysaltos(S)
information	sysconf(C) get system configuration	sysconf(C)
information	sysconf(S) get system configuration	sysconf(S)
	sysdef(M) output system definition	sysdef(M)
messages sys_nerr(S)	sys_errlist(S) errno(S) system error	sys_nerr(S)
information	sysfs(S) get file system type	sysfs(S)
system error messages	sys_nerr(S) sys_errlist(S) errno(S)	sys_nerr(S)
login(C) give you	system access	login(C)
acct(C) accounting	system	acct(C)
	system activity data collection	sadcon(M)
sar(C)	system activity report package	sar(C)
sar(M)	system activity report package	sar(M)
inir(M) clean the file	system and executes init	inir(M)
ckbupscd(M) check file	system backup schedule	ckbupscd(M)
stat(F) return data by stat	system call	stat(F)
intro(S) introduce	system calls, functions, and libraries	intro(S)
sysconf(C) get	system configuration information	sysconf(C)
sysconf(S) get	system configuration information	sysconf(S)
cu(C) call another UNIX	system	cu(C)
types(F) primitive	system data types	types(F)
fsdb(M) file	system debugger	fsdb(M)
sysdef(M) output	system definition	sysdef(M)
perror(S)	system error messages	perror(S)
sys_nerr(S) sys_errlist(S) errno(S)	system error messages	sys_nerr(S)

uuto(C) uupick(C) public UNIX-to-UNIX	system file copy	uuto(C)
mkunix(M) make bootable	system file with driver symbol table	mkunix(M)
mkunix(M) make bootable	system file with kernel symbol table	mkunix(M)
recover(C) restore contents of a file	system from tape	recover(C)
report information about a file	system fsinfo(M)	fsinfo(M)
help(C)	system help facility	help(C)
fstyp(M) determine the file	system identifier	fstyp(M)
dirent(F) file	system independent directory entry	dirent(F)
statfs(S) fstatfs(S) get file	system information	statfs(S)
brc(M)	system initialization procedure	brc(M)
lpadmin(M) configure the LP spooling	system	lpadmin(M)
mail(C)	system mail	mail(C)
menus(M) format of Business Shell menu	system	menus(M)
mkfs(M) construct a file	system	mkfs(M)
mount(S) mount a file	system	mount(S)
quot(C) summarize file	system ownership	quot(C)
rc0(M) commands to stop the operating	system	rc0(M)
reboot(C) automatically reboot the	system	reboot(C)
sysaltos(S) manufacturer specific	system requests	sysaltos(S)
reboot(S) shutdown or reboot the	system shutdn(S)	shutdn(S)
ustat(S) get file	system statistics	ustat(S)
fsstat(M) report file	system status	fsstat(M)
fstab(M) file	system table	fstab(M)
mnttab(M) mounted file	system table	mnttab(M)
asktime(C) set the	system time of day	asktime(C)
timezone(M) set default	system time zone	timezone(M)
archive(C) save a file	system to a streaming tape	archive(C)
shutdown(M) bring	system to single-user or shutdown	shutdown(M)
sysfs(S) get file	system type information	sysfs(S)
uname(S) get name of current UNIX	system	uname(S)
multiuser(C) singleuser(C) bring	system up multi/single-user mode	multiuser(C)
file transport program for uucp	system uucico(M)	uucico(M)
filesystem(M) format of a	system volume	filesystem(M)
who(C) display who is on the	system	who(C)
uutry(M) contact remote	system with debugging on	uutry(M)
volcopy(M) labelit(M) copy file	system with label checking	volcopy(M)
haltsys(C) close the file	systems and halt the CPU	haltsys(C)
digest(C) create menu	system(s) for the Business Shell	digest(C)
fsck(C) dfsck(C) check and repair file	systems	fsck(C)
	system(S) issue a shell command	system(S)
labelit(C) provide labels for file	systems	labelit(C)
umountall(C) mount/unmount multiple file	systems mountall(C)	mountall(C)
checklist(M) list file	systems processed by fsck	checklist(M)
bsearch(S) binary search of a sorted	table	bsearch(S)
retrieve symbol name for COFF symbol	table entry ldgetname(S)	ldgetname(S)
compute the index of a symbol	table entry of a COFF file ldtbindex(S)	ldtbindex(S)
ldtbread(S) read an indexed symbol	table entry of a COFF file	ldtbread(S)
syms(F) common object file symbol	table format	syms(F)
fstab(M) file system	table	fstab(M)
bootable system file with driver symbol	table mkunix(M) make	mkunix(M)
bootable system file with kernel symbol	table mkunix(M) make	mkunix(M)
mnttab(M) mounted file system	table	mnttab(M)
ldtbseek(S) seek to the symbol	table of a COFF file	ldtbseek(S)
setmnt(C) establish /etc/mnttab	table	setmnt(C)
hcreate(S) manage hash search	tables hsearch(S) hdestroy(S)	hsearch(S)
tabs(C) set	tabs on a terminal	tabs(C)
	tabs(C) set tabs on a terminal	tabs(C)

ctags(C) create a tags file _____ ctags(C)
tail(C) deliver the last part of a file _ tail(C)
sinh(S) cosh(S) tanh(S) hyperbolic functions _____ sinh(S)
functions trig(S) sin(S) cos(S) tan(S) asin(S) acos(S) trigonometric ____ trig(S)
save a file system to a streaming tape archive(C) _____ archive(C)
utility program for a streaming tape drive tapeutil(C) _____ tapeutil(C)
dump contents of a hard disk to tape dump.hd(C) _____ dump.hd(C)
frec(M) recover files from a back-up tape _____ frec(M)
restore contents of a file system from tape recover(C) _____ recover(C)
restore.hd(C) restore a hard disk from tape _____ restore.hd(C)
streaming tape drive tapeutil(C) utility program for a _____ tapeutil(C)
tar(C) archive files _____ tar(C)
trees tsearch(S) tfind(S) tdelete(S) twalk(S) manage binary search tsearch(S)
tee(C) create a tee in a pipe _____ tee(C)
tee(C) create a tee in a pipe _____ tee(C)
tk(C) paginator for Tektronix 4014 _____ tk(C)
reset(C) reset the teletype bit _____ reset(C)
directory operations directory(S) telldir(S) readdir(S) opendir(S) _____ directory(S)
file tmpnam(S) tempnam(S) create a name for a temporary tmpnam(S)
tmpfile(S) create a temporary file _____ tmpfile(S)
tmpnam(S) tempnam(S) create a name for a temporary file _____ tmpnam(S)
captoinfo(M) convert termcap to terminfo description _____ captoinfo(M)
termcap(M) terminal capability database _ termcap(M)
termcap(M) terminal capability database _____ termcap(M)
terminfo(M) terminal capability database _____ terminfo(M)
ct(C) spawn getty to a remote terminal _____ ct(C)
ctermid(S) generate file name for terminal _____ ctermid(S)
termio(M) general terminal interface _____ termio(M)
dial(S) establish an out-going terminal line connection _____ dial(S)
virtual terminal management _____ vt(M)
allow or disallow messages sent to a terminal mesg(C) _____ mesg(C)
getty(M) set terminal mode _____ getty(M)
tset(C) set terminal modes _____ tset(C)
clear(C) clear terminal screen _____ clear(C)
optimization package curses(S) terminal screen handling and _____ curses(S)
script(C) make a record of your terminal session _____ script(C)
gettydefs(M) speed and terminal settings used by getty _____ gettydefs(M)
tabs(C) set tabs on a terminal _____ tabs(C)
pscreen(C) set up terminal to print screen display _____ pscreen(C)
ttyname(S) isatty(S) find name of a terminal _____ ttyname(S)
discipline uugetty(M) set terminal type, modes, speed, line _____ uugetty(M)
ttys(M) login terminals file _____ ttys(M)
terminals(M) list of supported terminals _____ terminals(M)
term(M) conventional names for terminals _____ term(M)
terminals(M) list of supported terminals terminals(M)
kill(C) terminate a process _____ kill(C)
errstop(C) terminate error-logging demon _____ errstop(C)
exit(S) terminate process _____ exit(S)
wait for child process to stop or terminate wait(S) _____ wait(S)
query terminfo database _____ tput(C)
captoinfo(M) convert termcap to terminfo description _____ captoinfo(M)
infocmp(M) compare or print terminfo descriptions _____ infocmp(M)
tic(C) compile terminfo source _____ tic(C)
terminfo(M) terminal capability database terminfo(M)
termio(M) general terminal interface ____ termio(M)
term(M) conventional names for terminals term(M)
glossary(C) define common UNIX terms and symbols _____ glossary(C)

isnan(S) isnanf(S) isnand(S) test for floating point NaN ____ isnan(S)
test(C) evaluate an expression ____ test(C)
ed(C) red(C) invoke the ed text editor ____ ed(C)
edit(C) invoke the edit text editor ____ edit(C)
ex(C) invoke a text editor ____ ex(C)
diff(C) compare two text files ____ diff(C)
fspec(F) format specification in text files ____ fspec(F)
fmt(C) simple text formatter ____ fmt(C)
plock(S) lock process, text, or data in memory ____ plock(S)
binary search trees tsearch(S) tfind(S) tdelete(S) twalk(S) manage ____ tsearch(S)
tic(C) compile terminfo source ____ tic(C)
time(C) time a command ____ time(C)
clock(M) provide access to the time-of-day chip ____ clock(M)
cron(C) execute commands at specified times ____ cron(C)
time(S) get time ____ time(S)
touch(C) update access and modification times of a file ____ touch(C)
times(S) get process and child process times ____ times(S)
set file access and modification times utime(S) ____ utime(S)
times(S) get process and child process ____ times(S)
timezone(M) set default system time zone timezone(M)
tk(C) paginator for Tektronix 4014 ____ tk(C)
tmpfile(S) create a temporary file ____ tmpfile(S)
temporary file tmpnam(S) tempnam(S) create a name for a tmpnam(S)
characters conv(S) toupper(S) toascii(S) tolower(S) translate ____ conv(S)
popen(S) pclose(S) initiate pipe to/from a process ____ popen(S)
conv(S) toupper(S) toascii(S) tolower(S) translate characters ____ conv(S)
chkshlib(CP) tool for comparing shared libraries ____ chkshlib(CP)
tsort(C) sort a file topologically ____ tsort(C)
times of a file touch(C) update access and modification ____ touch(C)
translate characters conv(S) toupper(S) toascii(S) tolower(S) ____ conv(S)
query terminfo database tput(C) ____ tput(C)
tra(C) copy out a file as it grows ____ tra(C)
strace(M) print STREAMS trace messages ____ strace(M)
ptrace(S) process trace ____ ptrace(S)
aftp(C) transfer files between Altos machines ____ aftp(C)
conv(S) toupper(S) toascii(S) tolower(S) translate characters ____ conv(S)
tr(C) translate characters ____ tr(C)
xpd(M) transparent printer daemon ____ xpd(M)
uucico(M) file transport program for uucp system ____ uucico(M)
uusched(M) scheduler for the uucp file transport program ____ uusched(M)
tr(C) translate characters ____ tr(C)
ftw(S) walk a file tree ____ ftw(S)
tdelete(S) twalk(S) manage binary search trees tsearch(S) tfind(S) ____ tsearch(S)
trig(S) atan(S) atan2(S) trigonometric functions ____ trig(S)
sin(S) cos(S) tan(S) asin(S) acos(S) trigonometric functions trig(S) ____ trig(S)
functions trig(S) atan(S) atan2(S) trigonometric ____ trig(S)
acos(S) trigonometric functions trig(S) sin(S) cos(S) tan(S) asin(S) ____ trig(S)
true(C) return with a zero exit value ____ true(C)
manage binary search trees tsearch(S) tfind(S) tdelete(S) twalk(S) ____ tsearch(S)
tset(C) set terminal modes ____ tset(C)
tsort(C) sort a file topologically ____ tsort(C)
setmodem(C) set up tty port for a modem ____ setmodem(C)
makettys(M) create tty special files ____ makettys(M)
tty(C) get the current port name ____ tty(C)
terminal ttyname(S) isatty(S) find name of a ____ ttyname(S)
file of the current user ttyslot(S) find the slot in the utmp ____ ttyslot(S)
ttys(M) login terminals file ____ ttys(M)

tsearch(S) tfind(S) tdelete(S) twalk(S) manage binary search trees _____ tsearch(S)
dtype(C) determine disk type _____ dtype(C)
file(C) determine file type _____ file(C)
sysfs(S) get file system type information _____ sysfs(S)
uugetty(M) set terminal type, modes, speed, line discipline _____ uugetty(M)
types(F) primitive system data types _____ types(F)
types(F) primitive system data types ____ types(F)
date and time to string ctime(S) tzset(S) asctime(S) cftime(S) convert ___ ctime(S)
ua(C) user administration program _____ ua(C)
uadmin(S) administrative control _____ uadmin(S)
getpw(S) get name from UID _____ getpw(S)
ulimit(S) get and set user limits _____ ulimit(S)
umask(C) set file-creation mode mask ____ umask(C)
umask(S) set and get file creation mask _ umask(S)
systems mountall(C) umountall(C) mount/unmount multiple file mountall(C)
mount(C) umount(C) mount/unmount a file structure mount(C)
information uname(C) print the current UNIX _____ uname(C)
uname(S) get name of current UNIX system uname(S)
unget(CP) undo a previous get of an SCCS file _____ unget(CP)
file unget(CP) undo a previous get of an SCCS unget(CP)
stream ungetc(S) push character back into input ungetc(S)
uniq(C) report repeated lines in a file _ uniq(C)
mktemp(S) make a unique file name _____ mktemp(S)
constants unistd(F) file header for symbolic _____ unistd(F)
units(C) convert units _____ units(C)
units(C) convert units _____ units(C)
uname(C) print the current UNIX information _____ uname(C)
cu(C) call another UNIX system _____ cu(C)
glossary(C) define common UNIX terms and symbols _____ glossary(C)
uulog(C) uuname(C) copy files from UNIX to UNIX uucp(C) _____ uucp(C)
uuname(C) copy files from UNIX to UNIX uucp(C) uulog(C) _____ uucp(C)
uux(C) execute command on remote UNIX _____ uux(C)
uuto(C) uupick(C) public UNIX-to-UNIX system file copy _____ uuto(C)
link(M) unlink(M) link and unlink files and directories _____ link(M)
directories link(M) unlink(M) link and unlink files and _____ link(M)
unlink(S) remove directory entry _____ unlink(S)
pack(C) pcat(C) unpack(C) compress and expand files _____ pack(C)
pause(S) suspend process until signal _____ pause(S)
a file touch(C) update access and modification times of _ touch(C)
programs make(C) maintain, update, and regenerate groups of _____ make(C)
lsearch(S) lfind(S) linear search and update _____ lsearch(S)
sync(S) update super block _____ sync(S)
sync(C) update the super-block _____ sync(C)
upgrade.hd(C) upgrade an additional hard disk _____ upgrade.hd(C)
disk upgrade.hd(C) upgrade an additional hard upgrade.hd(C)
shutype(M) UPS shutdown configuration utility _____ shutype(M)
shuttype(S) get and set UPS shutdown limits _____ shuttype(S)
lint(CP) check C language usage and syntax _____ lint(CP)
du(C) summarize disk usage _____ du(C)
su(C) make the user a super-user or another user _____ su(C)
ua(C) user administration program _____ ua(C)
id(C) print user and group ID and names _____ id(C)
setuid(S) set user and group IDs _____ setuid(S)
crontab(C) manage user crontab files _____ crontab(C)
get character login name of the user cuserid(S) _____ cuserid(S)
environ(M) user environment _____ environ(M)
whoami(C) print effective current user id _____ whoami(C)

newgrp(C) log — user into a new group — newgrp(C)
ulimit(S) get and set — user limits — ulimit(S)
last(C) print last record of — user logins — last(C)
logname(S) return login name of — user — logname(S)
getuid(S) getegid(S) get real/effective — user or group IDs — getuid(S)
getuid(S) geteuid(S) get real/effective — user or group IDs — getuid(S)
getuid(S) getgid(S) get real/effective — user or group IDs — getuid(S)
make the user a super-user or another — user su(C) — su(C)
the slot in the utmp file of the current — user ttyslot(S) find — ttyslot(S)
write(C) write to another — user — write(C)
get and set maximum number of — users allowed to log in numusers(S) — numusers(S)
finger(C) find information about — users — finger(C)
wall(C) write to all — users — wall(C)
fuser(M) identify processes — using a file or file structure — fuser(M)
egrep(C) search file for pattern — using full regular expression — egrep(C)
ustat(S) get file system statistics — ustat(S)
cpset(C) install — utilities — cpset(C)
drive tapeutil(C) — utility program for a streaming tape — tapeutil(C)
setmode(C) printer modes — utility — setmode(C)
shutype(M) UPS shutdown configuration — utility — shutype(M)
modification times — utime(S) set file access and — utime(S)
utmp(M) wtmp(M) format of — utmp and wtmp entries — utmp(M)
utmpname(S) endutent(S) access — utmp file entry getut(S) getutent(S) — getut(S)
getut(S) setutent(S) getutline(S) access — utmp file entry — getut(S)
ttyslot(S) find the slot in the — utmp file of the current user — ttyslot(S)
entries — utmp(M) wtmp(M) format of utmp and wtmp — utmp(M)
entry getut(S) getutent(S) — utmpname(S) endutent(S) access utmp file — getut(S)
and permissions file — uucheck(M) check the uucp directories — uucheck(M)
uucp system — uucico(M) file transport program for — uucico(M)
cleanup — uucleanup(M) uucp spool directory — uucleanup(M)
uucheck(M) check the — uucp directories and permissions file — uucheck(M)
uusched(M) scheduler for the — uucp file transport program — uusched(M)
mail from — uucp link rmail(C) receives — rmail(C)
uucleanup(M) — uucp spool directory cleanup — uucleanup(M)
uustat(C) — uucp status inquiry and job control — uustat(C)
uucico(M) file transport program for — uucp system — uucico(M)
from UNIX to UNIX — uucp(C) uulog(C) uuname(C) copy files — uucp(C)
speed, line discipline — uugetty(M) set terminal type, modes, — uugetty(M)
to UNIX uucp(C) — uulog(C) uuname(C) copy files from UNIX — uucp(C)
uucp(C) uulog(C) — uuname(C) copy files from UNIX to UNIX — uucp(C)
file copy uuto(C) — uupick(C) public UNIX-to-UNIX system — uuto(C)
transport program — uusched(M) scheduler for the uucp file — uusched(M)
control — uustat(C) uucp status inquiry and job — uustat(C)
system file copy — uuto(C) uupick(C) public UNIX-to-UNIX — uuto(C)
debugging on — uutry(M) contact remote system with — uutry(M)
uux(C) execute command on remote UNIX — uux(C)
uuxqt(M) execute remote command requests — uuxqt(M)
val(CP) validate an SCCS file — val(CP)
val(CP) — validate an SCCS file — val(CP)
abs(S) return integer absolute — value — abs(S)
false(C) return with a nonzero exit — value — false(C)
getenv(S) return — value for environment name — getenv(S)
fabs(S) floor, ceiling, and absolute — value functions floor(S) ceil(S) — floor(S)
fmod(S) floor, ceiling, and absolute — value functions floor(S) — floor(S)
putenv(S) change or add — value to environment — putenv(S)
true(C) return with a zero exit — value — true(C)
values(F) machine-dependent — values — values(F)

	values(F) machine-dependent values	values(F)
vsprintf(S) print formatted output of	varargs list vprintf(S) vfprintf(S)	vprintf(S)
list	varargs(F) handles variable argument	varargs(F)
varargs(F) handles	variable argument list	varargs(F)
	vc(CP) version control	vc(CP)
get option letter from argument	vector getopt(S)	getopt(S)
assert(S)	verify program assertion	assert(S)
vc(CP)	version control	vc(CP)
get(CP) get a	version of an SCCS file	get(CP)
sccsdiff(CP) compare two	versions of an SCCS file	sccsdiff(CP)
output of varargs list vprintf(S)	vfprintf(S) vsprintf(S) print formatted	vprintf(S)
editor	vi(C) invoke a screen-oriented display	vi(C)
more(C)	view a file one full screen at a time	more(C)
	virtual terminal management	vt(M)
with label checking	volcopy(M) labelit(M) copy file system	volcopy(M)
filesystem(M) format of a system	volume	filesystem(M)
formatted output of varargs list	vprintf(S) vfprintf(S) vsprintf(S) print	vprintf(S)
varargs list vprintf(S) vfprintf(S)	vsprintf(S) print formatted output of	vprintf(S)
virtual terminal management	vt(M)	vt(M)
resource waitsem(S) nbwaitsem(S)	wait and check access to semaphore	waitsem(S)
wait(C)	wait completion of background processes	wait(C)
terminate wait(S)	wait for child process to stop or	wait(S)
processes	wait(C) wait completion of background	wait(C)
or terminate	wait(S) wait for child process to stop	wait(S)
access to semaphore resource	waitsem(S) nbwaitsem(S) wait and check	waitsem(S)
ftw(S)	walk a file tree	ftw(S)
	wall(C) write to all users	wall(C)
	wc(C) count lines, words, and characters	wc(C)
	what(C) identify files	what(C)
manual for program	whereis(C) locate source, binary, or	whereis(C)
id	whoami(C) print effective current user	whoami(C)
	who(C) display who is on the system	who(C)
	whodo(M) determine who is doing what	whodo(M)
users	whom(C) display in columns logged in	whom(C)
fold(C) fold long lines for finite	width output device	fold(C)
prof(F) profile	within a function	prof(F)
fgetc(S) getchar(S) get character or	word from a stream getc(S) getw(S)	getc(S)
putw(S) fputc(S) put character or	word on a stream putc(S) putchar(S)	putc(S)
wc(C) count lines,	words, and characters	wc(C)
cd(C) change	working directory	cd(C)
chdir(S) change	working directory	chdir(S)
getcwd(S) get path name of current	working directory	getcwd(S)
pwd(C) print	working directory name	pwd(C)
write(S)	write on a file	write(S)
putpwent(S)	write password file entry	putpwent(S)
wall(C)	write to all users	wall(C)
write(C)	write to another user	write(C)
	write(C) write to another user	write(C)
	write(S) write on a file	write(S)
open(S) open for reading or	writing	open(S)
drive(C) drive information	written during manufacturing	drive(C)
utmp(M) wtmp(M) format of utmp and	wtmp entries	utmp(M)
utmp(M)	wtmp(M) format of utmp and wtmp entries	utmp(M)
	xar(CP) maintain archives and libraries	xar(CP)
	xar(F) archive file format	xar(F)
	xargs(C) construct and execute commands	xargs(C)
	xcc(CP) invoke the XENIX compiler	xcc(CP)

enroll(C) xsend(C) xget(C) secret mail _____ enroll(C)
xld(CP) invoke the link editor _____ xld(CP)
from files xlist(S) fxlist(S) get name list entries xlist(S)
xnm(CP) print name list _____ xnm(CP)
adb(C) invoke x.out general purpose debugger _____ adb(C)
xpd(M) transparent printer daemon _____ xpd(M)
xref(CP) cross-reference C programs _____ xref(CP)
enroll(C) xsend(C) xget(C) secret mail _____ enroll(C)
xstr(CP) extract strings from C programs xstr(CP)
xtty(C) set the options for a port _____ xtty(C)
bessel(S) j0(S) y0(S) Bessel functions _____ bessel(S)
yacc(CP) invoke a compiler-compiler _____ yacc(CP)
yes(C) print a string repeatedly _____ yes(C)
true(C) return with a zero exit value _____ true(C)
timezone(M) set default system time zone _____ timezone(M)

Contents
Commands (C)

intro	Introduces operating system commands.
accept, reject	Allows/prevents print requests.
acct	Accounting system.
adb	Invokes a general purpose debugger (for x.out binaries).
add.hd	Adds an additional hard disk.
aftp	Transfers files between Altos machines.
archive	Saves the contents of a file system to a streaming tape drive.
asa	Interprets asa carriage control characters.
asktime	Sets the system time of day.
at, batch	Executes commands at a later time.
autoreboot	Automatically reboots the system.
awk	Pattern scanning and processing language.
badblock	Adds new bad sectors to the bad sector map.
banner	Prints large letters.
basename, dirname	Delivers portions of pathnames.
bc	Arbitrary-precision arithmetic language.
bdiff	Compares files too large for diff.
bfs	Scans big files.
bsh	Invokes the Business shell.
cal	Prints a calendar.
calendar	Invokes a reminder service.
cat	Concatenates and displays files.
cd	Changes working directory.
chmod	Changes the access permissions of a file or directory.
chown, chgrp	Changes owner or group ID.
chroot	Changes root directory for command.
clear	Clears terminal screen.
cmp	Compares two files.
comm	Selects or rejects lines common to two sorted files.
copy	Copies groups of files.
cp	Copies files.
cpio	Copies file archives in and out.

cpset	Installs utilities.
cron	Executes commands at specified times.
crontab	Manages user crontab files.
csh	Invokes a shell command interpreter with C-like syntax.
csplit	Splits files according to context.
ct	Spawns getty to a remote terminal.
ctags	Creates a tags file.
cu	Calls another UNIX system.
date	Prints and sets the date.
dc	Invokes an arbitrary precision calculator.
dd	Converts and copies a file.
devinfo	Displays device information.
devnm	Identifies device name on which files reside.
diff	Compares two text files.
diff3	Compares three files.
digest	Creates menu system(s) for the Business Shell.
dircmp	Compares directories.
disable	Disables logins on a port.
dos	Accesses MS-DOS files.
drive	Reads drive information written during manufacturing.
dtype	Determines disk type.
du	Summarizes disk usage.
dump.hd	Dumps the contents of a hard disk to tape.
echo	Echoes arguments.
ed, red	Invokes the ed text editor.
edit	Invokes the edit text editor (variant of ex).
egrep	Searches a file for a pattern using full regular expression.
enable	Enables logins on a port.
enroll, xsend, xget	Secret mail.
env	Sets environment for command execution.
errstop	Terminates error-logging demon.
ex	Invokes a text editor.
expr	Evaluates arguments as an expression.
factor	Factors a number.
false	Returns with a nonzero exit value.
fcopy	Copies a floppy diskette.
fdisk	Maintains disk partitions.
fgrep	Searches a file for a character string.
file	Determines file type.
find	Finds files.

finger	Finds information about users.
fleece	Looks for files in home directories.
fmt	Simple text formatter.
fold	Fold long lines for finite width output device.
format	Formats a floppy diskette.
from	Lists who my mail is from.
fsck, dfsck	Checks and repairs file systems.
getopt	Parses command options.
gets	Gets a string from the standard input.
glossary	Defines common UNIX terms and symbols.
graph	Draws a graph.
grep	Searches a file for a pattern.
haltsys	Closes out the file systems and halts the CPU.
hd	Displays files in hexadecimal format.
hdr	Displays selected parts of an object file.
head	Prints the first few lines of a stream.
help	Operating system help facility.
hplp, hplpR	Filters files for printing on an HP LaserJet printer.
id	Prints user and group ID and names.
ipcrm	Removes a message queue, semphore set, or shared memory id.
ipcs	Reports inter-process communication facilities status.
join	Joins two relations.
kill	Terminates a process.
killall	Kills all active processes.
labelit	Provides labels for file systems.
last	Prints last record of user and teletype logins.
leave	Reminds you when you have to leave.
line	Reads one line of input.
ln	Makes a link to a file.
login	Gives you to the system.
logname	Gets login name.
look	Finds lines in a sorted list.
lp, cancel	Sends/cancels requests to LP line printer.
lpenable, lpdisable	Enables/disables LP line printers.
lpr	Routes named files to printer spooler.
lpstat	Prints LP status information.

ls	Gives information about contents of directories.
mail	System mail.
make	Maintains, updates, and regenerates groups of programs.
mesg	Allows or disallows messages sent to a terminal.
mkdir	Makes a directory.
mknod	Builds special files.
mkstr	Creates an error message file from C source.
more	Views a file one full screen at a time.
mount, umount	Mounts/unmounts a file structure.
mountall, umountall	Mounts/unmounts multiple file systems.
multiuser, singleuser	Brings system up in multi-user/single-user mode.
mv	Moves or renames files and directories.
nawk	Pattern scanning and processing language.
newgrp	Logs user into a new group.
nice	Runs a command at a different priority.
nl	Adds line numbers to a file.
nohup	Runs a command immune to hangups and quits.
num	Numbers lines.
od	Displays files in octal format.
pack, pcat, unpack	Compresses and expands files.
passwd	Changes login password.
pconfig	Sets the port configuration.
pg	File perusal filter for CRTs.
pr	Prints files on the standard output.
printenv	Prints out the environment.
ps	Reports process status.
pscreen	Sets up terminal to print screen display.
pwd	Prints working directory name.
quot	Summarizes file system ownership.
random	Generates a random number.
reboot	Automatically reboots the system.
recover	Restores the contents of a file system from streaming tape to disk.
reset	Resets the teletype bits.
restore.hd	Restores a hard disk from tape.

rev	Reverse lines of a file.
rm, rmdir	Removes files or directories.
rmail	Receives mail (from uucp link).
sar	System activity reporter.
script	Makes a record of your terminal session.
sdb	Symbolic debugger.
sdiff	Compares files side-by-side.
sed	Invokes the stream editor.
see	Displays a file.
setmnt	Establishes /etc/mnttab table.
setmode	Port modes utility.
setmodem	Sets up tty port for use with a modem.
setpgrp	Executes a command in a new process group.
settime	Changes the access and modification dates of files.
sh, rsh	Invokes the shell command interpreter.
shl	Shell layer manager.
size	Prints section sizes of common object files.
sizefs	Determines the size of a logical disk drive.
sleep	Suspends execution for an interval.
sort	Sorts and merges files.
spell	Finds spelling errors.
spline	Interpolates smooth curves.
split	Splits a file into pieces.
ssp	Removes consecutive blank lines.
strings	Finds the printable strings in an object file.
stty	Sets the options for a port.
su	Makes the user a super-user or another user.
sum	Calculates checksum and counts blocks in a file.
swap	Changes swap device configuration.
sync	Updates the super-block.
sysconf	Gets system configuration information.
tabs	Set tabs on a terminal.
tail	Delivers the last part of a file.
tapeutil	Utility program for a streaming tape drive.
tar	Archives files.
tee	Creates a tee in a pipe.
test	Evaluates an expression.
tic	Compiles terminfo source.
time	Times a command.
tk	Paginator for Tektronix 4014.
touch	Updates access and modification times of a file.
tput	Queries terminfo database.

tr	Translates characters.
tra	Copies out a file as it grows.
true	Returns with a zero ("true") exit value.
tset	Sets terminal modes.
tsort	Sorts a file topologically.
tty	Gets the current port name.
ua	User Administration program.
umask	Sets file-creation mode mask.
uname	Displays the current operating system information.
uniq	Reports repeated lines in a file.
units	Converts units.
upgrade.hd	Upgrades an additional hard disk.
uucp, uulog, uuname	Copies files from UNIX to UNIX.
uustat	Uucp status inquiry and job control.
uuto, uupick	Public UNIX-to-UNIX system file copy.
uux	Executes command on remote UNIX.
vi	Invokes a screen-oriented display editor.
wait	Awaits completion of background processes.
wall	Writes to all users.
wc	Counts lines, words, and characters.
what	Identifies files.
whereis	Locates source, binary, or manual for program.
who	Displays who is on the system.
whoami	Prints effective current user id.
whom	Columnar display of system users.
write	Writes to another user.
xargs	Constructs and executes commands.
xtty	Sets the options for a port.
yes	Prints a string repeatedly.

Name

intro - Introduces operating system commands.

Description

This section describes use of the commands available in the Run-time System. Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option...*] [*cmdarg...*]

where:

name Is the name of an executable file.

option Is *-noargletter(s)* or, *-argletter<>optarg* (<> is optional whitespace).

- *noargletter* Is a single letter representing an option without an argument.
- *argletter* Is a single letter representing an option requiring an argument.
- *optarg* Is an argument (character string) satisfying the preceding *argletter.*

cmdarg Is a pathname (or other command argument) not beginning with -. By itself, - indicates the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. **Getopts**(C) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.

2. Command names must include only lower-case letters and digits.

3. Option names (*option* above) must be one character long.

4. All options must be preceded by "-".

5. Options with no arguments may be grouped after a single "-".

6. The first option-argument (*optarg* above) following an option must be preceded by white space.

7. Option-arguments cannot be optional.

8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., **-o xxx,z,yy** or **-o "xxx z yy"**).

9. All options must precede operands (*cmdarg* above) on the command line.

10. "--" may be used to indicate the end of the options.

11. The order of the options relative to one another should not matter.

12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.

13. "-" preceded and followed by white space should only be used to mean standard input.

See Also

getopts(C), getopt(S)

Diagnostics

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

Notes

Not all commands adhere to the syntax described here.

(BLANK)

Name

accept, reject - Allows/prevents print requests to a line-printer or class of printers.

Syntax

/usr/lib/accept *destination...*
/usr/lib/reject [**-r** [*reason*]] *destination...*

Description

Accept allows **lp**(C) to accept requests for the named destinations. A destination can be either a printer or a class of printers. Use **lpstat**(C) to find the status of destinations.

Reject prevents **lp**(C) from accepting requests for the named destinations. A destination can be either a printer or a class of printers. Use **lpstat**(C) to find the status of destinations.

These commands can only be used by the super-user.

You can use the following option with **reject:**

-r[*reason*] Associates a reason that prevents **lp** from accepting requests. This reason applies to all printers mentioned up to the next **-r** option. *Reason* is reported by **lp** when users direct requests to the named destinations. *Reason* is also reported by **lpstat.** If **-r** is not present or is given without a reason, a default reason will be used.

Files

/usr/spool/lp/*

See Also

enable(C), lp(C), lpadmin(M), lpinit(M), lpsched(M), lpstat(C)

Name

acct - Accounting system.

Description

The accounting system, contained in the directory /usr/lib/acct, provides ways to collect per-process resource utilization data, record connect sessions, monitor disk use, and charge fees to specific logins. The accounting system has a set of C language programs and shell procedures to reduce the accounting data into summary files.

For a description of the accounting system, see the *Operations Guide*.

Files

/usr/lib/acct/*	C programs and shell procedures to run the accounting system
/usr/adm/*	Active data collection files
/usr/adm/acct/nite/*	Files reused daily by runacct
/usr/adm/acct/sum/*	Cumulative summary files updated by runacct
/usr/adm/acct/fiscal/*	Periodic summary files created by monacct

See Also

/etc/init, /etc/rc2, /etc/wtmp, /usr/lib/cron

Name

adb - Invokes a general-purpose debugger.

Syntax

adb [-w] [-p *prompt*] [*objfile* [*corefile*]]

Description

Adb is a general purpose debugging program for use only with x.out binaries. **Adb** may be used to examine files and to provide a controlled environment for the execution of programs. To debug COFF programs, use **sdb**(C).

Objfile 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 *objfile* is a.out. *Corefile* is assumed to be a core image file produced after executing *objfile*; the default for *corefile* is core.

Requests to **adb** are read from the standard input and responses are written to the standard output. The options are:

-w Both *objfile* and *corefile* are created if necessary and opened for reading and writing so that files can be modified using **adb**. The Break/Del key causes **adb** to return to the next command.

-p Defines the prompt string: any combination of characters. The default is an asterisk (*).

In general requests to **adb** are of the form:

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

If *address* is present, then the current address (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* is a special expression having the form:

[*segment*:]*offset*

where *segment* gives the address of a specific text or data segment, and *offset* gives an offset from the beginning of that segment. If *segment* is not given, the last segment value given in a command is used.

The interpretation of an *address* depends on the context it is used in. If a sub-process 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 the following section, "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 # or 0x; otherwise a decimal number.

integer.fraction
A 32-bit floating point number.

'*cccc*' The ASCII value of up to 4 characters. The backslash (\) 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 *corefile*. The register names are eax, ebx, ecx, edx, edi, esi, ebp, esp, efi, eip, cs, ds, es, fs, gs, and ss. The name fl refers to the status flags.

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 *objfile*. An initial underscore (_) or tilde character (~) will be prepended to *symbol* if needed.

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

(*exp*) The value of the expression *exp*.

Monadic Operators

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

-*exp* Integer negation.

~*exp* Bitwise complement.

Dyadic Operators

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

e1+e2 Integer addition (+) or subtraction (-).

*e1*e2* Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1\e2 Bitwise disjunction.

e1^e2 Remainder after division of *e1* by *e2*.

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" following for further details.)

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

/f Locations starting at *address* in *corefile* 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, the last format is used. The format letters available are:

o	2	Prints 1 word in octal. All octal numbers output by **adb** are preceded by 0.
O	4	Prints 2 words in octal.
q	2	Prints in signed octal.
Q	4	Prints long signed octal.
d	2	Prints in decimal.
D	4	Prints long decimal.
x	2	Prints 1 word in hexadecimal.
X	4	Prints 2 words in hexadecimal.
u	2	Prints as an unsigned decimal number.
U	4	Prints long unsigned decimal.
f	4	Prints the 32-bit value as a floating point number.
F	8	Prints double floating point.
b	1	Prints the addressed byte in octal.
c	1	Prints the addressed character.
C	1	Prints the addressed character using the following escape convention. Character values 000 to 040 are printed as an at-sign (@) followed by the corresponding character in the octal range 0100 to 0140. The at-sign character itself is printed as @@.

s	*n*	Prints the addressed characters until a zero character is reached.
S	*n*	Prints a string using the at-sign (@) escape convention. Here *n* is the length of the string including its zero terminator.
Y	4	Prints 4 bytes in date format (see **ctime**(S)).
i	*n*	Prints as machine 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	Prints 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
A	0	Prints the value of dot in absolute form.
p	2	Prints 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	Prints a space.
n	0	Prints a newline.
"..."	0	Prints the enclosed string.
^		Decrements dot by the current increment. Nothing is printed.
+		Increments dot by 1. Nothing is printed.
-		Decrements dot by 1. Nothing is printed.

Available commands include:

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

[?/][**Ll**] *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. If a question mark (?) is given, a text segment is affected; if a slash (/), a data segment.

[?/][Ww] *value ...*
Writes the 2-byte *value* into the addressed location. If the command is W, writes 4 bytes. Odd addresses are not allowed when writing to the subprocess address space. If a question mark (?) is given, a text segment is affected; if a slash (/), a data segment.

[?/][Mm] *segnum fpos size*
Sets new values for the given segment's file position and size. If *size* is not given, then only the file position is changed. The *segnum* must the segment number of a segment already in the memory map (see "Addresses"). If a question mark (?) is given, a text segment is affected; if a slash (/), a data segment.

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

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

$modifier

Miscellaneous commands. The available modifiers are:

<*f* Reads commands from the file *f* and returns.

>*f* Sends output to the file *f*, which is created if it does not exist.

r Prints the general registers and the instruction addressed by *ip*. Dot is set to *ip*.

f Prints the floating registers in single or double length.

b Prints all breakpoints and their associated counts and commands.

c C stack backtrace. If *address* is given, it is taken as the address of the current frame (instead of bp). 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 Prints the names and values of external variables.

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

s Sets the limit for symbol matches to *address* (default 255).

o Sets input and output default format to octal.

d Sets input and output default format to decimal.

x Sets input and output default format to hexadecimal.

q Exits from **adb**.

v Prints all non-zero variables in octal.

m Prints the address map.

:modifier

Manage a subprocess. Available modifiers are:

brc Sets a breakpoint at *address*; 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.

dl Delete a breakpoint at *address*.

r [*arguments*]
Runs *objfile* 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.

R [*arguments*]
Same as the **r** command except that *arguments* are passed through a shell before being passed to the program. This means shell metacharacters can be used in filenames.

cos The subprocess is continued and signal *s* is passed to it (see **signal**(S)). If *address* is given, then 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 **co** except that the subprocess is single-stepped *count* times. If there is no current subprocess, then *objfile* 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 *corefile*. If *corefile* does not appear to be a core file then these values are set from *objfile*:

b The base address of the data segment.

d The data segment size.

e The entry point.

m The execution type.

n The number of segments.

s The stack segment size.

t The text segment size.

Addresses

Addresses in **adb** refer to either a location in a file or in actual memory. When there is no current process in memory, **adb** addresses are computed as file locations, and requested text and data are read from the *objfile* and *corefile* files. When there is a process, such as after a **:r** command, addresses are computed as actual memory locations.

All text and data segments in a program have associated memory map entries. Each entry has a unique segment number. In addition, each entry has the file position of that segment's first byte, and the physical size of the segment in the file. When a process is running, a segment's entry has a virtual size which defines the size

of the segment in memory at the current time. This size can change during execution.

When an address is given and no process is running, the file location corresponding to the address is calculated as:

effective-file-address = file-position + offset

If a process is running, the memory location is simply the *offset* in the given segment. These addresses are valid if and only if:

0 <= *offset* <= *size*

where *size* is physical size for file locations and virtual size for memory locations. Otherwise, the requested address is not legal.

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, file position is set to 0, and *size* is set to the maximum file size. In this way, the whole file can be examined with no address translation.

All appropriate values are kept as signed 32-bit integers so that **adb** may be used on large files.

Files

/dev/swap
a.out
core

See Also

ptrace(S), a.out(F), core(F) in the *Reference (CP, S, F) Programmer's Guide*

Diagnostics

The message "adb" appears when there is no current command or format.

Comments appear when there are inaccessible files, syntax errors, abnormal termination of commands, etc.

Exit status is 0, unless the last command failed or returned non-zero status.

Notes

A breakpoint set at the entry point is not effective on initial entry to the program.

System calls cannot be single-stepped.

Local variables whose names are the same as an external variable may foul up the accessing of the external.

Name

add.hd - Adds an additional hard disk.

Syntax

add.hd [-d] [2] [3]

Description

The **add.hd** command is a shell script that installs an additional hard disk if your system supports more than one add-on hard disk. You must be the super-user to use this command.

If you don't specify the number (i.e., 2 or 3) on the command line, the script prompts you for the number. Once you reply with a correct number, the system installs the additional hard disk.

To safely add (or read) a hard disk, log in as **root**, and follow these steps:

1. Reboot the system.

2. Enter system maintenance mode.

3. Immediately execute **add.hd.** Do not open the drive in any way (e.g., by executing **mount, swap,** etc.) before you add the hard disk.

Add.hd runs the **layout**(C) program, which divides the disk into the following areas:

- Spare sectors for bad spots (non-SCSI drives only).

- File system.

- Extra swap space. **Add.hd** asks you if you want a swap area on this drive. If you do, answer yes. **Add.hd** will prompt you for the size of the area. If you answer no, no swap area is created on the drive.

Next the **badblock**(C) program checks the additional drive for bad spots. If there are any, it maps them into the spare area. When **badblock** is finished (takes 15 - 20 minutes), you are asked to specify the number of inodes or press **Retn** for the default.

Before the additional hard disk goes to multiuser mode, it is checked, and if necessary **fsck**(C) is run. Then, the add-on hard disk is mounted.

The directory used for the add-on hard disk is /usr2 for the second hard disk, /usr3 for the third hard disk. The add-on hard disk remains mounted as /usr2 (/usr3) when **add.hd** exits and whenever the system is in multiuser mode.

-d Non-destructive add. Does not run **layout, badblock,** and **mkfs** routines. Does not destroy the files on the disk. The **-d** option makes the /usr*n* directory and the necessary devices in /dev, and adds the **mount**(C) command to /etc/fstab, if it does not already exist.

See Also

layout(C), make.hd(C), sizefs(C), upgrade.hd(C)

Name

aftp - Transfers files between Altos machines.

Syntax

aftp [-f *device*] [-s *speed*] [*file...*]

Description

The **aftp** program allows you to transfer files between two Altos computer systems.

The **aftp** program must be run on both the sending and receiving computer. The port that **aftp** is running on must have login disabled (see **disable**(C)). Either side may be started first, but both sides must be started within about 1 minute of each other. The sending side will output 's' every few seconds until communication is established with the other side; likewise, the receiving side will output 'w' every few seconds. During file transfer, **aftp** will output a '*' every time a 128 byte block is successfully transmitted, and a '?' every time a block is retransmitted to overcome a transmission error.

For information on setting up your systems see the *Operations Guide.*

Options

-f *device*	The special file *device* is used to transfer files between the machines. The ports associated with the devices on each machine should be connected via a null modem cable.
-s *speed*	The transmission rate is set to *speed.* Currently supported speeds are 1200, 2400, 4800, and 9600 bits per second. The default transmission rate is 9600 baud.

file On the sending side, *file* is a file or a list of files If *file* is "-", standard input is sent. On the receiving side, *file* is an existing directory into which the files are received. If *file* is omitted, files are received into the current directory. If *file* is "--", received files are written to standard output.

CP/M and MP/M Systems

The **aftp** program is compatible with the **ftp** program available for Altos CP/M and MP/M systems, so files can be transferred between CP/M-MP/M systems and UNIX systems. Files sent to MP/M and CP/M systems must have file names that are legal on those systems. Files sent from MP/M and CP/M systems to UNIX systems may end up with file names containing and sometimes ending with spaces; the UNIX shells can deal with these file names if the entire name is enclosed in double quotes.

Since MP/M and CP/M pad files with Ctrl-z (octal 32), Ctrl-z is deleted from the end of files sent to UNIX systems. The files also contain Ctrl-m entries (octal 15), which need to be stripped out once on the UNIX side. For example,

cat *file1* | **tr -d '\015'** > *file2*

Name

archive - Saves the contents of a file system to a streaming tape drive.

Syntax

archive [-A][-e][-i *string*][-V] *file_system mag_tape_device*

Description

You must be the super-user to use the **archive** command.

Use **archive** to copy the contents of a file system (specified by *file_system*) to a cartridge tape (specified by *mag_tape*). If the files will not fit on a single tape, you will be prompted to install a new tape.

The system must be in single-user (maintenance) mode when you back up /dev/root. Be sure to specify *mag_tape_device* as /dev/rsct. If possible, run **archive** on an unmounted file system.

If you are backing up the first hard disk (/dev/hd0b), use the **dump.hd**(C) command.

Options

-A Aborts the backup (at the end of the tape) when a write error occurs.

-e Erases the tape prior to writing data on it. We recommend you use the -e option before you back up your files.

-i Puts *string* (any string up to 128 characters) into the header block on the tape.

-V Verifies that the contents of the tape match the contents of the disk (bit-for-bit compare).

Examples

This command backs up the first hard disk to tape.

```
/etc/dump.hd  Retn
```

The **dump.hd**(C) command calls **archive**, which gives the appropriate parameters for the first hard disk.

To restore the first hard disk from tape, boot from the diskette labeled "Root File System" and select option **c** on the menu.

For example, this command backs up the second hard disk to tape.

```
/etc/umount /dev/hd1b  Retn
archive /dev/rhd1b /dev/rsct  Retn
```

This command restores the second hard disk from tape.

```
/etc/umount /dev/hd1b  Retn
recover /dev/rsct /dev/rhd1b  Retn
```

You can check the device name by typing

```
mount  Retn
```

The screen displays the device name, for example

```
/dev/hd1b on /usr2 read/write on Mon Mar 31 10:17:12 1986
```

See Also

recover(C), dump.hd(C), restore.hd(C)
Operations Guide

Name

asa - Interprets asa carriage control characters.

Syntax

asa [**-s**] [*file* ...]

Description

Asa processes the output of Fortran programs that use asa carriage control characters. **Asa** processes the files whose names are given as arguments, or standard input if no file names are given, and sends the results to the standard output.

The first character of each line is interpreted as an asa control character as follows:

' '	Single space before printing
'0'	Double space before printing
'-'	Triple space before printing
'1'	New page before printing
'+'	Overstrike the previous line

If the first character of a line is not one those listed above, it is treated as if it were a space. **Asa** forces the first line of each file to begin on a new page.

The **-s** option suppresses error messages from **asa.**

Asa returns one of the following values as its exit status:

0	No error
-1	Output error
>0	Some input files could not be opened; the return code is the total number of files that could not be opened.

Name

asktime - Sets the system time of day.

Syntax

/etc/asktime

Description

The **asktime** command prompts you for the date: year, month, day, and time: hour and minute. This command synchronizes the real-time clock and the system clock. You must be the super-user to use this command.

Example

This example sets the new time, date, and year to "9:23 January 1, 1987".

```
Current System Time is Wed Nov 3 14:36:23 1986
Enter date (yymmdd) or press RETURN if ok:  870101
Enter time (hhmm) or press RETURN if ok:  0923
```

See Also

date(C)

Name

at, batch - Executes commands at a later time.

Syntax

at *time* [*date*] [+ *increment*]
at -r *job...*
at -l [*job...*]
batch

Description

At and **batch** read commands from standard input to be executed at a later time. At allows you to specify when the commands should be executed, while jobs queued with **batch** will execute when system load level permits. At may be used with the following options:

-r Removes jobs previously scheduled with **at.**

-l Reports all jobs scheduled for the invoking user.

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

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

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

indicate GMT. The special names **noon, midnight, now,** and **next** are also recognized.

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

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

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

At and **batch** write the job number and schedule time to standard error. **Batch** submits a batch job. It is almost equivalent to "at now", but not quite. For one, it goes into a different queue. For another, "at now" will respond with the message "too late."

At -r removes jobs previously scheduled by **at** or **batch.** The job number is the number given to you previously by the **at** or **batch** command. You can also get job numbers by typing **at -l.** You can only remove your own jobs unless you are the super-user.

Examples

The **at** and **batch** commands read from standard input the commands to be executed at a later time. **Sh**(C) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output. This sequence can be used at a terminal:

```
batch
sort filename > outfile
Ctrl-d
```

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

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

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

echo "sh shellfile" | at 1900 thursday next week

Files

/usr/lib/cron	Main cron directory
/usr/lib/cron/at.allow	List of allowed users
/usr/lib/cron/at.deny	List of denied users
/usr/lib/cron/queuedefs	Scheduling information
/usr/spool/cron/atjobs	Spool area

See Also

kill(C), mail(C), nice(C), ps(C), sh(C), sort(C), cron(C)

Diagnostics

Complains about various syntax errors and times out of range.

Name

autoreboot - Toggles the autoreboot process on and off. (Series 2000 only)

Syntax

/etc/autoreboot [on | off]

Description

When **autoreboot** is enabled (on), rebooting the machine requires no user input from the console. Thus, the system will reboot without the attendance of a system administrator. **Autoreboot off** disables the process.

Autoreboot with no options displays whether the autoreboot process is "on" or "off."

Normally, when the machine is recovering from a system crash or a power loss with no UPS installed, the rebooting process will invoke **fsck**(C) (file-system check) which will wait for user response.

Enabling **autoreboot** causes **fsck -y** to be run instead, which asks no questions. The output of the **fsck** command is saved and mailed to root after the **fsck** is finished. The message "Redirecting fsck output..." is printed when the file-system check begins.

Name

awk - Invokes a pattern processing editor.

Syntax

awk [**-Fc**] [**-f** *file*] ['*prog*'] [*file*]

Description

The **awk** command scans each input file for lines that match any of a set of patterns specified in *prog*. With each pattern in a program 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*. The *prog* string should be enclosed in single quotation marks (') to protect it from the shell. The **-Fc** option uses *c* as a field separator.

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 spaces. 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*; *condition*; *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 delimited with double quotes (").

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**(S)).

The built-in function **length** returns the length of its argument taken as a string, or of the whole line if no argument is given. 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**(S) 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**(C). 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 the first 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 **-F***c* option.

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

Examples

Print lines longer than 72 characters:

```
awk 'length > 72'
```

Print first two fields in opposite order:

```
awk '{ print $2, $1 }'
```

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

```
awk '/start/, /stop/'
```

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

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

See Also

nawk(C), grep(C), lex(CP), sed(C)

Notes

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

Name

badblock - Adds new bad sectors to the bad sector map.

Syntax

badblock -p *disk_no*
badblock [-v] [-n] -u *minor_dev block_no ...*
badblock [-v] -i *disk_no*

Description

The **badblock** command displays the bad sector list and map, adds bad sectors to the bad sector list and map, or initializes the bad sector map from the bad sector list. This command can only be executed by the super user.

-i Initializes the hard disk bad sector map from the bad sector list for the specified hard disk (*disk_no*). This operation is executed by the system when the disk is initialized.

CAUTION

The -i option must not be used after you have added any bad blocks with the -u option unless you are completely rebuilding the disk. Doing so will cause bad sectors to be mapped incorrectly.

For the Series 500, if you partition the disk with more than one partition, the *disk_no* is a 2-digit number. The first digit is the physical disk number (0 or 1). The second digit is the partition number (0, 1, 2, or 3).

When initializing the bad sector map (**-i** option) on a Series 500, **badblock** will create the bad sector list by scanning the disk for bad sectors.

-n No attempt is made to copy the bad block to the newly mapped sector.

-p Displays the bad sector list and bad sector map of the specified disk number (*disk_no*).

For the Series 500, see the **-i** option.

-u *minor_dev*
Allows new bad blocks to be added to the bad sector list and map. When a bad sector is found on a disk, an error message indicating the major/minor device number and bad block number is printed on the console. The two parameters required for the **-u** option are given in this error message. The minor device number (*minor_dev*) is given in the message in the form (*major_dev/minor_dev*). Use the **badblock** utility to copy the bad block to the newly mapped sector:

badblock -u *minor_dev block_no*

To check that the bad block was mapped, type:

badblock -p *disk_no*

-v Lists the bad sectors map on the screen.

Badblock will make 10 attempts to copy the sector before it gives up, and reports the success or failure of the copy. The user will be prompted before the new bad sector map and new bad sector list are actually written to the disk. The new bad sector map information will take effect immediately after the user permits the write of the new map.

Note

The **-i** and **-p** options are not supported on SCSI hard drives.

Name

banner - Prints large letters.

Syntax

banner *string* ...

Description

Banner prints its arguments (each up to ten characters long per line) in large letters on the standard output. This is useful for printing names at the front of printouts.

See Also

echo(C)

Name

basename, dirname - Delivers portions of path names.

Syntax

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

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 (``) within shell procedures.

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

Examples

The following example, invoked with the argument **/usr/src/cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

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

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

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

See Also

sh(C)

Name

bc - Arbitrary-precision arithmetic language.

Syntax

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

Description

Bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The **bc**(C) utility is actually a preprocessor for **dc**(C), which it invokes automatically unless the **-c** option is present. In this case the **dc** input is sent to the standard output instead.

Bc may also be used as a desktop calculator. The following example shows the sequence necessary to set the radix to base-64 and convert the decimal value 884 into its corresponding base-64 representation.

```
bc  Retn
obase=64  Retn
884  Retn
13 52          (response from bc)
```

The options are as follows:

-c Compile only. The output is sent to the standard output.

-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 semi-colons or new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of **dc**(C). 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.

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.

Files

/usr/lib/lib.b	Mathematical library
/usr/bin/dc	Desk calculator proper

See Also

dc(C)

Notes

The **bc** command does not yet recognize the logical operators, && and ||. The **for** statement must have all three expressions (E's). **Quit** is interpreted when read, not when executed.

Name

bdiff - Compares files too large for diff.

Syntax

bdiff *file1 file2* [*n*] [-s]

Description

Bdiff compares two files, finds lines that are different, and prints them on the standard output. It allows processing of files that are too large for **diff**(C). **Bdiff** splits each file into *n*-line segments, beginning with the first nonmatching lines, and invokes **diff** on the corresponding segments. If both arguments are specified, they must appear in the order indicated above. The arguments are:

n The number of lines into which **bdiff** splits each file for processing. The default value is 3500. This is useful when 3500-line segments are too large for **diff**(C) causing it to fail. If the optional third argument is given, and it is numeric, it is used as the value for *n*.

-s Suppresses printing of **bdiff** diagnostics. Note that this does not suppress printing of diagnostics from **diff**(C), which **bdiff** calls.

If *file1* (or *file2*) is a dash (-), the standard input is read.

The output of **bdiff** is exactly like that of **diff.** Line numbers are adjusted to account for the segmenting of the files, and the output looks 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(C), help(C)

Name

bfs - Big file scanner.

Syntax

bfs [-] *name*

Description

The **bfs** command is (almost) like **ed**(C) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). **Bfs** is usually more efficient than **ed**(C) for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where **csplit**(C) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional - suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed, as in **ed**(C). Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under **ed**(C) are supported. In addition, regular expressions may be surrounded with two symbols besides / and **?**: **>** indicates downward search without wrap-around, and **<** indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

The **e, g, v, k, p, q, w, =, !** and null commands operate as described under **ed**(C). Commands such as **---, +++-, +++=, -12,** and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no *remembered* file name. The **w** command is independent of output diversion, truncation, or crunching (see the **xo, xt** and **xc** commands, below).

The following additional commands are available:

xf *file* Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the **xf**. The **xf** commands may be nested to a depth of 10.

xn List the marks currently in use (marks are set by the **k** command).

xo [*file*] Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666 (readable and writable by everyone), unless your *umask* setting (see **umask**(C)) dictates otherwise. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label* This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the : and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(.,.)**xb**/*regular expression*/*label*
A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, **.** is set to the line matched and a jump is made to *label*. This command is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command:

xb/^/*label*

is an unconditional jump. The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt *number*

Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

xv[*digit*][*spaces*][*value*]

The variable name is the specified *digit* following the **xv**. The commands **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. The command **xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6**:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

```
g/%5/p
```

would globally search for the characters **100** and print each line containing a match. To escape the special meaning of **%**, a \ must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing **printf**(S) of characters, decimal integers, or strings.

Another feature of the **xv** command is that the first line of output from a system command can be stored into a variable. The only requirement is that the first character of *value* be an **!**.

For example:

```
w junk
xv5!cat junk
rm junk
echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a \.

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*
xbn *label* These two commands will test the last saved *return code* from the execution of a system command (*!command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string *size*.

```
xv55
:1
/size/
xv5!expr%5 - 1
!if 0%5 != 0 exit 2
xbn 1
xv45
:1
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz 1
```

xc [*switch*]
If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it is not. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

See Also

csplit(C), ed(C), umask(C), more(C)

Diagnostics

There is a ? for errors in commands, if prompting is turned off. Self-explanatory error messages are produced when prompting is on.

Name

bsh - Invokes the Business shell.

Syntax

bsh [-fhqs] [*menusystem*]

Description

The Business shell is a menu-driven command language interpreter. It may be installed as the "login shell" in the password file, or invoked directly by typing **bsh.**

The **bsh** command is implemented using the termcap and curses facilities. Run **bsh** from a terminal defined in /etc/termcap. If the terminal is not defined in /etc/termcap, **bsh** will be aborted.

The **bsh** command should only be run interactively, not in the background.

Options

-f Starts **bsh** in fast mode. In this mode, a prompt whose first letter is lower-case alphabetic character is executed immediately when the first letter is typed. The system does not wait for a terminating newline. Prompts whose first letter is upper-case alphabetic wait for a terminating newline before executing the requested actions. Fast mode is the default mode, if not overridden by the command line or the BSHINIT variable (see below). The current mode may be changed during execution through use of the **?mode** command (described below).

-h Displays a short help message describing how to invoke **bsh.**

-q Displays a one line description of the syntax used to invoke **bsh.**

-s Start **bsh** in slow mode. In this mode, all prompts must be terminated by a newline before execution occurs. The current mode may be changed during execution through use of the **?mode** command (described below).

If you write your own menu system, **bsh** utilizes the designated *menusystem* instead of the standard one (/etc/menusys.bin). Prior to use by **bsh**, a menu system must be "digested" using the digest utility. If it is not, or if it is not read-accessible, **bsh** issues an error message and terminates.

Using the Business Shell

Prompts

Typing any of the prompts on the current menu screen immediately causes the actions associated with the prompt to be executed. Selecting a prompt with no associated action causes an error message to be displayed.

An action may be any one of the following:

- Go to a specified menu
- Execute a **sh** script
- Execute a **bsh** internal command

Menu Name

Typing the name of a menu causes it to immediately become the current menu. If the menu name is misspelled, or if it does not exist in the current menu system, an error message is displayed.

New Line

Typing a newline causes the immediately preceding menu to become the current one. If there is no previous menu, an error message is displayed. The **bsh** command does not distinguish between **Line Feed** and **Retn** -- both generate a newline.

?

Typing a question mark (?) causes the "help" menu associated with the current menu to be displayed.

??

Typing a pair of question marks (??) causes the **bsh** system help files to be displayed. It contains much the same information as is presented here.

Menu Name?

Typing the name of a menu followed by question mark calls up the designated help menu to become the current one.

!*command*

The exclamation point (!) allows you to escape to the standard shell (**sh**). The command must follow the usual rules as described in the **sh** documentation. In particular, the command may consist of a sequence of shell commands separated by semicolons - thus several actions may be invoked. If the command is absent, sh is invoked as a sub-shell with no arguments. In this case, **bsh** will be resumed as soon as the sub-shell terminates. (Usually, this is accomplished by sending the sub-shell an end-of-file message. End-of-file is Ctrl-d on most terminals.)

?index

This special command causes **bsh** to display its internal "index" for the current menu system. The index contains the names of every accessible menu.

?mode

This special command allows you to change from "slow" mode to "fast" mode and vice versa. You are asked if you wish to change to the alternate mode. If your response begins with "y" or "Y", the change is made, otherwise the current mode remains in effect.

Delete

The **bsh** command immediately returns to the top-level command interpreter upon receipt of an interrupt signal. Such a signal is usually generated via the Break/Delete or **Rubout** key.

backspace

The **bsh** command understands the backspace function as obtained from /etc/termcap.

Escape, Cancel

The **bsh** command interprets the Esc or **Cancel** key to mean "re-start input." Ctrl-x on other terminals also performs this function.

Ctrl-r

If you cannot clear the screen, you can force **bsh** to clear it and redisplay the current contents by pressing Ctrl-r.

q, Q, Quit

Typing a **q, Q** or **Quit** all have the same effect: **bsh** is terminated. If **bsh** is your login shell, the use of this command also results in your being logged out.

Environment

The **BSHINIT** environment variable contains the initial value of the default mode ("fast" or "slow"). If this variable does not exist in the environment, **bsh** assumes "fast" mode. BSHINIT should be set by inserting the line BSHINIT=fast or BSHINIT=slow into your .profile file.

Note that even if **bsh** is designated as the "login shell" in /etc/passwd, your .profile file will be interpreted correctly (see **login**(C) and **sh**(C)). In particular, any overriding definitions you may have for the kill and erase characters will be correctly interpreted by **bsh.**

Files

/.profile	Contains commands to be executed during login
/etc/menusys.bin	Default menu system used by bsh
/etc/passwd	Used to define a user's login name, password, home directory, shell
/etc/termcap	Contains terminal attribute descriptions

See Also

login(C), sh(C), termcap(M)

(BLANK)

Name

cal - Prints a calendar.

Syntax

cal [[*month*] *year*]

Description

Cal prints a calendar for the specified *year.* If *month* is also specified, a calendar for that month only is printed. If no arguments are specified, a calendar for the current month is printed. *Year* must be a number between 1 and 9999; *month* must be a number between 1 and 12.

Notes

Note that "cal 84" refers to the year 84, not 1984.

The calendar produced is that for England and her colonies. Note that England switched from the Julian to the Gregorian calendar in September of 1752, at which time eleven days were excised from the year. To see the result of this switch, try **cal 9 1752.**

Name

calendar - Invokes a reminder service.

Syntax

calendar [-]

Description

Calendar looks at the file named calendar in the user's current directory, and prints (to the user's terminal) lines that contain today's or tomorrow's date. Month-day dates such as "Sep. 7," "september 7", and "9/7", are recognized, but not "7 September", "7/9" or "07/09" for September 7.

On weekends, "tomorrow" extends through Monday. Lines that contain the date of a Monday will be sent to the user on the previous Friday. This is not true for holidays.

When an argument is present, **calendar** does its job for every user who has a calendar file in his login directory and sends the user the results by **mail**(C). Normally this is done daily, in the early morning, under the control of **cron**(C).

Files

calendar	
/usr/lib/calprog	To figure out today's and tomorrow's dates
/etc/passwd	
/tmp/cal*	
/usr/lib/crontab	

See Also

cron(C), mail(C)

Notes

To get reminder service, a user's calendar file must have read permission for all.

Name

cat - Concatenates and prints files.

Syntax

cat [-u] [-s] [-v [-t] [-e]] *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 in the third.

If no input file is given, or if the argument - is encountered, **cat** reads from the standard input.

The following options apply to **cat.**

-u The output is not buffered. (The default is buffered output.)

-s Cat is silent about non-existent files.

-v Causes non-printing characters (with the exception of tabs, new-lines, and form-feeds) to be printed visibly. Control characters are printed ^X (control-*x*); the DEL character (octal 0177) is printed ^?. Non-ASCII characters (with the high bit set) are printed as M-*x*, where *x* is the character specified by the seven low-order bits.

When used with the -v option, the following options may be used.

-t Causes tabs to be printed as ^I's.

-e Causes a $ character to be printed at the end of each line (prior to the new-line).

The -t and -e options are ignored if the -v option is not specified.

Notes

Command formats such as

cat *file1 file2 >file1*

will cause the original data in *file1* to be lost; therefore, take care when using shell special characters.

See Also

cp(C), pg(C), pr(C)

Name

cd - Changes directory.

Syntax

cd [*directory*]

Description

Use the **cd** command to change directories. Typing **cd** with no argument places you in your login (home) directory. This command is built into the shells; it is not a separate command.

Examples

This command moves you up one level of your directory.

```
cd ..
```

This command changes the current directory to /usr/wendy/memos/meetings.

```
cd /usr/wendy/memos/meetings
```

This command moves you into the April directory, which is a subdirectory of the Letters directory.

```
cd Letters/April
```

Related Commands

pwd(C), sh(C)

Name

chmod - Changes the access permissions of a file or directory.

Syntax

chmod *mode file...*
chmod *mode directory...*

Description

The permissions of the named *files* or *directories* are changed according to *mode*, which may be symbolic or absolute. Absolute changes to permissions are stated using octal numbers:

chmod *nnn file*

where *n* is a number from 0 to 7. Symbolic changes are stated using mnemonic characters:

chmod *a operator b file*

where *a* is one or more characters corresponding to **user**, **group**, or **other**; where *operator* is +, -, and =, signifying assignment of permissions; and where *b* is one or more characters corresponding to type of permission.

An absolute mode is given as 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 is turned on ((see **chmod**(S))
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

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions themselves.

Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	**Group**	**Other**
rwx	rwx	rwx

This example (meaning that user, group, and others all have reading, writing, and execution permission to a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Thus, to change the mode of a file's (or directory's) permissions using **chmod**'s symbolic method, use the following syntax for *mode*:

[*who*] *operator* [*permission(s)*],...

A command line using the symbolic method would appear as follows:

chmod g+rw *file*

This command would make *file* readable and writable by the group.

The *who* part can be stated as one or more of the following letters:

u user's permissions
g group's permissions
o others permissions

The letter **a** (all) is equivalent to **ugo** and is the default if *who* is omitted.

Operator can be + to add *permission* to the file's mode, - to take away *permission*, or = to assign *permission* absolutely. (Unlike other symbolic operations, = has an absolute effect in that it resets all other bits.) Omitting *permission* is only useful with = to take away all permissions.

Permission is any compatible combination of the following letters:

r	reading permission
w	writing permission
x	execution permission
s	user or group set-ID is turned on
t	sticky bit is turned on

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter **s** is only meaningful with **u** or **g**, and **t** only works with **u**.

Only the owner of a file or directory (or the super-user) may change a file's mode. Only the super-user may set the sticky bit. In order to turn on a file's set-group-ID, your own group ID must correspond to the file's, and group execution must be set.

Examples

chmod a-x *file*

chmod 444 *file*

The first example denies execution permission to all. The absolute (octal) example permits only reading permissions.

chmod go+rw *file*

chmod 666 *file*

These examples make a file readable and writable by the group and others.

chmod =rwx,g+s *file*

chmod 2777 *file*

These last two examples enable all to read, write, and execute the file; and they turn on the set-group-ID.

Notes

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the Remote File Sharing manual.

See Also

ls(C) and chmod(S) in the *Reference (CP, S, F)*

Name

chown, chgrp - Changes owner or group.

Syntax

chown *owner file...*
chown *owner directory...*
chgrp *group file...*
chgrp *group directory...*

Description

Chown changes the owner of the *files* or *directories* to *owner*. The owner may be either a decimal user ID or a login name found in the password file. **Chgrp** changes the group ID of the *files* or *directories* to *group*. The group may be either a decimal group ID or a group name found in the group file. If either command is invoked by other than the super-user, the set-user-ID and setgroup-ID bits of the file mode, 04000 and 02000 respectively, will be cleared. Only the owner of a file (or the super-user) may change the owner or group of that file.

Files

/etc/passwd
/etc/group

Notes

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls -l** command leads you to believe. For more information see the Remote File Sharing manual.

See Also

chmod(C), group(M), passwd(M) and chown(S) in the *Reference (CP, S, F)*

Name

chroot - Changes the root directory for a command.

Syntax

/etc/chroot *newroot command*

Description

Chroot causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the path names is changed for the command and any of its child processes to *newroot.* Furthermore, upon execution, the initial working directory is *newroot.*

Notice, however, that if you redirect the output of the command to a file:

chroot *newroot command >x*

will create the file *x* relative to the original root of the command, not the new one.

The new root path name is always relative to the current root: even if a **chroot** is currently in effect, the *newroot* argument is relative to the current root of the running process.

This command can be run only by the super-user.

See Also

cd(C) and chroot(S) in the *Reference (CP, S, F)*

Notes

One should exercise extreme caution when referencing device files in the new root file system.

Name

clear - Clears terminal screen.

Syntax

clear

Description

Clear clears your screen. It looks in the environment for the terminal type and then in /etc/lib/terminfo to figure out how to clear the screen.

Files

/etc/lib/terminfo Terminal information data base

Name

cmp - Compares two files.

Syntax

cmp [**-l**] [**-s**] *file1 file2*

Description

Cmp compares two files and, if they are different, displays the byte and line numbers of the differences. If *file1* is -, the standard input is used.

The options are:

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

-s Returns an exit code only, 0 for identical files, 1 for different files, and 2 for an inaccessible or missing file.

This command should be used to compare binary files; use **diff**(C) or **diff3**(C) to compare text files.

See Also

comm(C), diff(C), diff3(C)

Diagnostics

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

Name

comm - Selects or rejects lines common to two sorted files.

Syntax

comm [- [**123**]] *file1 file2*

Description

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see **sort**(C)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. A - for *file1* means the standard input.

The **1**, **2**, or **3** flags 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** doesn't work and will print nothing.

See Also

cmp(C), diff(C), sort(C), uniq(C)

Name

copy - Copies groups of files.

Syntax

copy [*option*] ... *source* ... *dest*

Description

The **copy** command copies the contents of directories to another directory. It is possible to copy whole file systems since directories are made when needed.

If files, directories, or special files do not exist at the destination, then they are created with the same modes and flags as the source. In addition, the super-user may set the user and group ID. The owner and mode are not changed if the destination file exists. Note that there may be more than one source directory. If so, the effect is the same as if the **copy** command had been issued for each source directory with the the same destination directory for each copy.

All of the options must be given as separate arguments and they may appear in any order even after the other arguments. The *options* are:

-a Asks the user before attempting a copy. If the response does not begin with a "y" for yes, then a copy is not done. This option also sets the **-ad** option.

-l Uses links whenever they can be used. Otherwise a copy is done. Note that links are never used for special files or directories.

-n Requires the destination file to be new. If not, then the **copy** command does not change the destination file. The **-n** flag is meaningless for directories. For special files, an **-n** flag is assumed (i.e., the destination of a special file must not exist).

-o If set, then every file copied has its owner and group set to those of the source. If not set, then the file's owner is the user who invoked the program.

-m If set, then every file copied has its modification time and access time set to that of the source. If not set, then the modification time is set to the time of the copy.

-r If set, then every directory is recursively examined as it is encountered. If not set, then any directories that are found are ignored.

-ad Asks the user whether an **-r** flag applies when a directory is discovered. If the answer does not begin with a "y," then the directory is ignored.

-v If the verbose option is set, messages are printed that reveal what the program is doing. *source* This may be a file, directory or special file. It must exist. If it is not a directory, then the results of the command are the same as for the **cp**(C) command.

dest

The destination must be either a file or directory that is different from the source.

If *source* and *destination* are anything but directories, then **copy** acts just like a **cp** command. If both are directories, then **copy** copies each file into the destination directory according to the flags that have been set.

Notes

Special device files can be copied. When they are copied, any data associated with the specified device is not copied.

Name

cp - Copies files.

Syntax

cp *file1 file2*
cp *file directory*
cp -r *directory1 directory2*

Description

Use the **cp** command to make a copy of a file within the same directory or to copy a file from one directory to another. In the latter case you can either rename the file or keep the same name.

The **-r** option recursively copies directory trees.

NOTE

You cannot copy a directory into a file.

Examples

This command makes a copy of the file letter1 and renames it letter2.

cp letter1 letter2

This command places a copy of file letter1 in the directory January.

cp letter1 January/letter1

Related Commands

ln(C), mv(C), rm(C)

Name

cpio - Copies file archives in and out.

Syntax

cpio -o [**aBcvV**] [**-C***bufsize*] [**-M***message*] <*name-list* >*collection*
cpio -o [**aBcvV**] **-O***collection* [**-C***bufsize*] [**-M***message*] <*name-list*
cpio -i [**bBcdfkmrsStuvV6**] [**-C***bufsize*] [**-M***message*] [*pattern*] <*collection*
cpio -i [**bBcdfkmrsStuvV6**] **-I***collection* [**-C***bufsize*] [**-M***message*] [*pattern*]
cpio -p [**adlmruvV**] *directory* <*name-list*

Description

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

Cpio -i (copy in) extracts from the standard input (which is assumed to be the product of a previous **cpio -o**) the names of files selected by zero or more *patterns* given in the name-generating notation of **sh**(C). In patterns, the special characters ?, *, and [...] match the slash (/) character. The default for *patterns* is * (i.e., select all files).

Remember to escape special characters to prevent expansion by the shell.

Cpio -p (pass) copies out and in during a single operation. Destination pathnames are interpreted relative to the named *directory*.

The meanings of the available option flags are:

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

-b Swaps both bytes and halfwords. Use only with the -i option.

-B Blocks input/output 512 bytes to the record (does not apply to the pass option; meaningful only with data directed to or from raw devices).

-c Writes header information in ASCII character form for portability.

-C*bufsize*
Sets buffer size to *bufsize.*

-d Directories are created as needed.

-f Do not consider patterns on the command line.

-I*collection*
Reads input from *collection* instead of standard input.

-k In case of I/O errors in reading, tries several times before reporting I/O error and exiting. If the file header is corrupted, prints a message about the situation and continues reading input.

-l Whenever possible, links files rather than copying them. Usable only with the **-p** (pass) option.

-m Retains previous file modification time. This option is ineffective on directories that are being copied.

-M*message*
Sets alternate message *message* for end-of-media.

-O*collection*
Writes output to *collection* instead of standard output.

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

-s Swaps bytes. Use only with the **-i** option.

-S Swaps halfwords. Use only with the **-i** option.

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

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

-v Verbose. Causes a list of filenames to be printed. When used with the -t option, the table of contents looks like the output of an ls -l command (see ls(C)).

-V Verbose. Prints a dot (.) for each file.

-6 Processes an old file. Use only with the -i (copy in) option.

Examples

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

```
ls | cpio -o >/dev/fd0

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

or:

```
find . -print | cpio -oB >/dev/rfd0
```

See Also

ar(CP), cpio(F), find(C)

Notes

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

Name

cpset - Installs utilities.

Syntax

/etc/cpset [**-o**] *filename destination* [*mode*] [*owner*] [*group*]

Description

The **cpset** command is used by make files to install new utilities. You can also invoke this command from the command line.

Options

-o	If *filename* already exists, **cpset** moves it to OLD*filename*.
filename	The name of the utility you want to install.
destination	The destination directory of the utility.
mode	Permissions for the utility (default is set by **umask**(C)).
owner group	The user and group id for the installed utility.

See Also

chgrp(C), chmod(C), chown(C), make(C), umask(C)

Name

cron - Executes commands at specified dates and times.

Syntax

/etc/cron

Description

Cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in crontab files (in the /usr/spool/cron/crontabs directory). Users can submit their own crontab file by using the **crontab**(C) command. 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/rc2.d/S??cron.

The file 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 (0-6 with 0=Sunday). Each of these patterns may contain a number in the range above, two numbers separated by a minus means a range (inclusive); a list of numbers separated by commas means 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.

Cron only examines crontab files during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Examples

A sample crontab file follows:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /usr/lib/atrun
0,10,20,30,40,50 * * * * /etc/dmesg ->>/usr/adm/messages
1,21,41 * * * * (echo -n ' ';  date;  echo ) >/dev/console
```

Cron creates log entries in /usr/lib/cron/log. Be sure to monitor the size of /usr/lib/cron/log so that it doesn't unreasonably consume disk space.

Files

/usr/lib/cron	Main cron directory
/usr/lib/cron/log	Accounting information
/usr/spool/cron	Spool area

See Also

crontab(C), sh(C), init(M)

Name

crontab - User crontab file.

Syntax

crontab [*file*]
crontab -r
crontab -l

Description

Crontab copies the specified file, or standard input, if no file is specified, into a directory that holds all users' crontabs. The **-r** option removes a user's crontab from the crontab directory. The **-l** option lists your own **crontab** file.

Users are permitted to use **crontab** if their names appear in the file /usr/lib/cron/cron.allow. If that file does not exist, the file /usr/lib/cron/cron.deny is checked to determine if the user should be denied access to **crontab.** If neither file exists, only root can submit a job. If either file is **at.deny**, global usage is permitted. The allow/deny files consist of one user name per line.

A **crontab** file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0-59)
hour (0-23)
day of the month (1-31)
month of the year (1-12)
day of the week (0-6 with 0=Sunday)

Each of these patterns may be either an asterisk (that is, all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (that is, an inclusive range). Two fields specify days (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 runs a command on the first and fifteenth of each month, as well as on every Monday.

To specify days by only one field, set the other fields to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a **crontab** file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by a \) 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.

The shell is invoked from your $HOME directory with an *arg0* of **sh.** Users who desire to have their .profile executed must explicitly do so in the crontab file. **Cron** supplies a default environment for every shell, defining HOME, LOGNAME, SHELL(=/bin/sh), and PATH(=:/bin:/usr/bin).

Files

/usr/lib/cron	Main cron directory
/usr/spool/cron/crontabs	Spool area
/usr/lib/cron/log	Accounting information
/usr/lib/cron/cron.allow	List of allowed users
/usr/lib/cron/cron.deny	List of denied users

See Also

sh(C), cron(C)

Notes

Remember to redirect the standard output and standard error of commands! If not, any generated output or errors will be mailed to you.

If you inadvertently enter the **crontab** command with no argument(s), do not attempt to get out with a Ctrl-d. This will cause all entries in your crontab file to be removed. Instead, exit with a Break/Del.

Name

csh - Invokes a shell command interpreter with C-like syntax.

Syntax

csh [**-cefinstvVxX**] [*arg* ...]

Description

The **csh** command, a command language interpreter, begins by executing commands from the file **.cshrc** in the user's home directory. If this is a login shell, **csh** also executes commands from the file **.login** in the user's home directory. The shell will then begin reading commands from the terminal, prompting with **%**.

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 user's 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 the |, <, or > characters are doubled (||, <<, or >>), these pairs form single words. The parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with a backslash (\). A new line 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. Within pairs of \ or " characters, a new line 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 does not have this special meaning when preceded by a \ and placed inside the quotation marks `, ', or ".

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 a semi-colon (;), and are then executed sequentially. A sequence of pipeline commands may be executed without waiting for them to terminate by following it with an &. Such a sequence is automatically prevented from being terminated by a hangup signal; the **nohup** command need not be used.

Any of the above may be placed in parentheses 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.")

Substitutions

The following sections describe the various transformations the shell performs on the input in the order in which they occur.

History Substitutions

History substitutions can be used to reintroduce sequences of words from previous commands, possibly performing modifications on these words. Thus history substitutions provide a generalization of a redo function.

History substitutions begin with the character ! and may begin anywhere in the input stream if a history substitution is not already in progress. This ! may be preceded by a \ to prevent its special meaning; a ! is passed un-

changed 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 size of which is controlled by the *history* variable. The previous command is always retained. Commands are numbered sequentially from 1.

For example, 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 a ! 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 !w 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. The form !# references the current command (the one being typed in). It allows a word to be selected from further left in the line, to avoid retyping a long name, as in !#:1.

To select words from an event we can follow the event specification by a : and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, and so on. 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 one 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 a sequence of modifiers can be placed, each preceded by a :. The following modifiers are defined:

h Removes a trailing pathname component

r Removes a trailing **.xxx** component

s/*l*/*r*/
Substitutes *l* for *r*

t Removes all leading pathname components

& Repeats the previous substitution

g Applies the change globally, prefixing the above

p Prints the new command but does not execute it

q Quotes the substituted words, preventing substitutions

x Like **q**, but breaks into words at blanks, tabs, and newlines

Unless preceded by a **g**, the modification is applied only to the first modifiable word. In any case it is an error for no word to be applicable.

The left 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 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 nonblank 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 that 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 variable and command expansion may occur.

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**. 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 that paginates its arguments to the lineprinter.

Variable Substitution

The shell maintains a set of variables, each of which has as a 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 that causes command input to be echoed. The setting of this variable results from the -v command line option.

Other operations treat variables numerically. The at-sign (@) command permits numeric calculations to be performed and the result assigned to a variable. However, variable values are 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 dollar sign ($) characters. This expansion can be prevented by preceding the dollar sign with a backslash (\) except within double quotation marks (") where it always occurs, and within single quotation marks (') where it never occurs. Strings quoted by back quotation marks (`) are interpreted later (see "Command Substitution" below) so dollar sign substitution does not occur there until later, if at all. A dollar sign is passed unchanged if followed by a blank, tab, or end-of-line.

Input and 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 generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

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

The following sequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable that 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, digits, and underscores.

  If *name* is not a shell variable, but is set in the environment, then that value is returned; however, 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 variable's 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. Only one : modifier is allowed 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.

$$ Substitutes the (decimal) process number of the (parent) shell.

Command and Filename Substitution

Command and filename substitution are applied selectively to the arguments of built-in 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 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 back quotation marks. 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 double quotation marks, 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 filenames which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing filename, 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, it expands to the invoker's 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 the user's 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 unchanged.

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/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/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 unchanged.

Input/Output

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

< *name*	Opens file *name* (which is first variable, command and filename expanded) as the standard input.
<< *word*	Reads the shell input up to a line which is identical to *word.* The variable *word* is not subjected to variable, filename, or command substitution, and each input line is compared to *word* before any substitutions are made on this input line. Unless a quoting backslash, double, or single quotation mark, or a back quotation mark appears in *word,* variable and command substitution is performed on the intervening lines, allowing \ to quote $, \, and `. Commands that are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resulting 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, it is truncated, and its previous contents is lost.

If the variable *noclobber* is set, then the file must not already exist or it must 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 to 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 >.

If a command is run detached (followed by &), then the default standard input for the command is the empty file /dev/null. Otherwise the 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. Use the << mechanism to present inline data, so shell command scripts can function as components of pipelines and the shell can block-read its input.

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

Expressions

A number of the built-in commands (to be described later) 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, with the operators:

```
== and !=
<=, >=, <, and >
<< and >>
+ and -
* / and %
```

forming groups at the same level. The == and != operators compare their arguments as strings; all others operate on numbers. 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. No two components of an expression can appear in the same word; they should be surrounded by spaces except when adjacent to components of expressions which are syntactically significant to the parser (& | < > ( )).

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, then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible,

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 control command files (shell scripts) and, in limited but useful ways, 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 commands will succeed on nonseekable inputs.)

Built-In Commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, then it is executed in a subshell. The following list describes the syntax and function of the built-in commands:

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. The value for *name* is not allowed to be *alias* or *unalias*.

break Causes execution to resume after the **end** of the nearest enclosing **foreach** or **while** statement. 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* Changes the shell's working directory to directory *name.* If no argument is given, changes 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 Continues 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.

echo *wordlist* The specified words are written to the shell's standard output. A **\c** causes the echo to complete without printing a newline. A **\n** in *wordlist* causes a newline to be printed. Otherwise, the words are echoed, separated by spaces.

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

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).

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 built-in command **continue** may be used to continue the loop prematurely and the built-in command break to terminate it prematurely. When this command is read from the terminal, the loop is read once prompting with **?** before any statements in the loop are executed.

glob *wordlist* — Like **echo** but no \ escapes are recognized and words are delimited by null characters in the output. Useful for programs that 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.

history Displays the history event list.

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. The value for *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.

if (*expr*) **then**
...
else if (*expr2*) **then**
...
else
...
endif
If the specified *expr* is true, the commands to the first **then** are executed; else if *expr2* is true then the commands to the second **then** 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-then** clause must appear alone on its input line or after an **else.**)

logout Terminates a login shell. The only way to log out if *ignoreeof* is set.

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* ...."

The *command* is always executed in a subshell, and the restrictions placed 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. Unless the shell is running detached, **nohup** has no effect. All processes detached with **&** are automatically run with **nohup.** (Thus, **nohup** is not really needed.)

onintr
onintr -
onintr *label* Controls the action of the shell on interrupts. The first form restores the default action of the shell on interrupts: 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.

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 redirection occurs 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 that 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 the environment variable *name* to be **value**, a single string. Useful environment variables are TERM, the type of your terminal and SHELL, the shell you are using.

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.

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 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 match the **unalias** *pattern*.

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

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.

wait
: All child processes are waited for. If the shell is interactive, an interrupt can disrupt the wait, at which time the shell prints names and process numbers of all children known to be outstanding.

while (*expr*)
...
end
: While the specified expression evaluates non-zero, the commands between **while** and the matching **end** are evaluated. Use **break** and **continue** to terminate or continue the loop prematurely. **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.

@
@ *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 |, 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.

Assignment operators, such as *= and +=, are available as in C. The space separating the name from the assignment operator is optional. Spaces are mandatory in separating components of *expr* which would otherwise be single words.

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

Predefined Variables

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

The shell copies the environment variable PATH into the variable *path*, and copies the value back into the environment whenever *path* is set. Thus 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.

argv Set to the arguments of the shell, from this variable positional parameters are substituted, i.e., $1 is replaced by $*argv*[1].

cdpath Gives a list of alternate directories searched to find subdirectories in **cd** commands.

child The process number printed when the last command was forked with **&**. This variable is unset when this process terminates.

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 nonbuilt-in commands, all expansions occur before echoing. Built-in commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars Can be assigned a two-character string. The first character is used as a history character in place of **!**, the second character is used in place of the **^** substitution mechanism. For example, set histchars = ",;" will cause the history characters to be comma and semicolon.

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

home The home directory of the user, initialized from the environment. The filename expansion of **~** refers to this variable.

ignoreeof If set, the shell ignores end-of-file from input devices that are terminals. This prevents a shell from accidentally being terminated by typing a **Ctrl-d.**

mail The files where the shell checks for mail. This is done after each command completion and will result in a prompt, if a specified interval has elapsed. The shell sends the message, "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, the shell sends the message "New mail in *name*" when there is mail in the file *name*.

noclobber As described in the section "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 that 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.

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, only full pathnames will execute. The usual search path is /bin, /usr/bin, and ., but this may vary from system to system. For the super-user the default search path is /etc, /bin and /usr/bin. A shell that 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 use 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 super-user).

shell The file in which the shell resides. This is used in forking shells to interpret files that have execute bits set, but which are not executable by the system. (See the section "Nonbuilt-In 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. Abnormal termination results in a core dump. Built-in commands that fail return exit status **1**, all other built-in commands set status **0**.

time Controls automatic timing of commands. If set, any command that takes more than this many CPU seconds will cause a line giving user, system, and real times and a utilization percentage (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.

Nonbuilt-In Command Execution

When a command to be executed is found to not be a built-in command, the shell attempts to execute the command via **exec**(S). 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 (with **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 pathname 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 **cd** 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 and 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 pathname 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 nonzero exit status.

-f The shell will start faster, because it will neither search for nor execute commands from the file .cshrc in the user's 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 Causes the *echo* variable to be set even before .cshrc is executed.

After the flag arguments have been processed, if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options were 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 on a typical system most shell scripts are written for the standard shell (see **sh**(C)), the C shell will execute such a standard shell if the first character of a script is not a **#** (if the script does not start with a comment). Remaining arguments initialize the variable *argv*.

Signal Handling

The shell normally ignores *quit* signals. The *interrupt* and *quit* signals are ignored for an invoked command if the command is followed by &; otherwise the signals have the values that the shell inherited from its parent. The shell's handling of interrupts 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.

Files

/etc/default/.cshrc	Read by each shell at the beginning of execution.
/etc/default/.login	Read by login shell, after .cshrc at login.
/etc/default/.logout	Read by login shell, at logout.
/bin/sh	Shell for scripts not starting with a **#**.
/tmp/sh*	Temporary file for <<.
/dev/null	Source of empty file.
/etc/passwd	Source of home directories for name.
/etc/cshrc	Default file of automatically invoked commands.

Limitations

Words can be no longer than 512 characters. The number of arguments to a command which involves filename expansion is limited to 1/6 number of characters allowed in an argument list, which is 10240, less the characters in the environment. Also, 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

access(S), a.out(F), environ(M), exec(S), fork(S), pipe(S), signal(S), umask(S), wait(S)
User's Guide

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Built-in control structure commands like **foreach** and **while** cannot be used with |, &, or ;.

Commands within loops, prompted for by ?, are not placed in the history list.

It is not possible to use the colon (:) modifiers on the output of command substitutions.

Csh attempts to import and export the PATH variable for use with regular shell scripts. This only works for simple cases, where the PATH contains no command characters.

This version of **csh** does not support or use the process control features of the 4th Berkeley Distribution.

Name

csplit - Splits files according to context.

Syntax

csplit [-f *prefix*] [-k] [-s] *file arg1* [... *argn*]

Description

Csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1...argn*. By default the sections are placed in xx00...xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*.

01: From the line referenced by *arg1* up to the line referenced by *arg2*

.
.
.

n: From the line referenced by *argn* to the end of file.

The options are:

-f *prefix* If the -f option is used, the created files are named *prefix*00...*prefix*n. The default is xx00...xx*n*.

-k Csplit normally removes created files if an error occurs. If this option is present, csplit leaves previously created files intact.

-s Csplit normally prints the character counts for each file created. If this option is present, csplit suppresses the printing of all character counts.

The arguments (*arg1...argn*) can be a combination of the following:

/rexp/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or - some number of lines (e.g., /Page/-5).

%rexp% This argument is the same as /rexp/, except that no file is created for the section.

lnno A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{num} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* arguments that contain blanks or other characters meaningful to the shell in the appropriate quotation marks. Regular expressions may not contain embedded newline characters. **Csplit** does not affect the original file; it is the user's responsibility to remove it.

Examples

csplit -f cobol file '/procedure division/' /par5./ /par16./

This example creates four files, cobol00...cobol03. After editing the "split" files, they can be recombined as follows:

cat cobol0[0-3] > file

Note that this example overwrites the original file.

csplit -k file 100 {99}

This example would split the file at every 100 lines, up to 10,000 lines. The -k option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

Assuming that *prog*.c follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in *prog*.c.

See Also

ed(C), regex(S), sh(C)

Diagnostics

Self-explanatory, except for:

arg - out of range

which means that the given argument did not reference a line between the current position and the end of the file.

Name

ct - Spawns getty to a remote terminal.

Syntax

ct [-h] [-v] [-w*n*] [-x*n*] [-s*speed*] *telno* ...

Description

Ct dials the phone number of a modem that is attached to a terminal, and spawns a **getty** process to that terminal. *Telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. If you specify more than one telephone number, **ct** tries each in succession until one answers; this is useful for specifying alternate dialing paths.

Ct tries each line in the file /usr/lib/uucp/Devices until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, **ct** asks if it should wait for one, and if so, for how many minutes it should wait before it gives up. **Ct** continues to try to open the dialers at one-minute intervals until the specified limit is exceeded.

Options

-x*n* Produces a detailed output of the program execution on stderr (used for debugging). The debugging level, *n*, is a single digit; -x9 is the most useful value.

-w*n* Overrides dialogue. *n* is the maximum number of minutes that **ct** is to wait for a line.

-h Prevents **ct** from hanging up current line to allow that line to answer the incoming call (the default is to hang up). Waits for the termination of the specified **ct** process before returning control to the user's terminal.

-v Sends a running narrative to the standard error output stream.

-s Sets the data rate; *speed* is the baud rate. The default rate is 1200.

If there is already an active **getty**(M) or **uugetty**(M) running on the dial-out port prior to invoking **ct**, **ct** will establish the connection and exit, allowing the running **getty**(M) or **uugetty**(M) to print the "login:" prompt on the destination terminal. The connection is terminated after the user on the destination terminal logs out.

On the other hand, if the dial-out port is disabled, **ct** spawns a new **getty**(M) on the port after establishing connection, and waits for the user on the destination terminal to log out.

After the user logs out, **ct** prompts,

```
Reconnect?
```

If the response begins with the letter **n** or there is no response in 20 seconds, the line is dropped; otherwise, **getty**(M) will be started again and the **login:** prompt will be printed.

To log out properly, type Ctrl-d.

Of course, the destination terminal must be attached to a modem that can answer the telephone.

Files

/usr/lib/uucp/Devices
/usr/adm/ctlog

See Also

cu(C), login(C), uucp(C), getty(M), uugetty(M)

Name

ctags - Creates a tags file.

Syntax

ctags [**-u**] [**-w**] [**-x**] *name* ...

Description

Ctags makes a tags file for **vi**(C) from the specified C sources. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition. These are given in separate fields on the line, separated by blanks or tabs. Using the tags file, **vi** can quickly find these function definitions. Options are:

-u Causes 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. (Note: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the tags file.)

-w Suppresses warning diagnostics.

-x Produces a list of function 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. With the **-x** option no tags file is created. This is a simple index which can be printed out as an off-line readable function 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.

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(C), vi(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Name

cu - Calls another UNIX system.

Syntax

cu [-s*speed*] [-l*line*] [-h] [-t] [-d] [-o | -e] [-n] *telno*
cu [-s *speed*] [-h] [-d] [-o | -e] -l *line*
cu [-h] [-d] [-o | -e] *systemname*

Description

Cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files.

NOTE

Before you run **cu**, set up the serial port as: **xtty ixon ixoff iflow oflow.** See **xtty**(C) for an explanation of these flags.

Cu accepts the following options and arguments:

-s*speed* Specifies the transmission speed (300, 1200, 2400, 4800, 9600). The default value is "Any" speed which will depend on the order of the lines in the /usr/lib/uucp/Devices file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

-l*line* Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the -l option is used without the -s option, the speed of a line is taken from the Devices file. When the -l and -s options are both used together, **cu** will search the Devices file to check if the requested speed for the re-

quested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., /dev/tty*nn*) in which case a telephone number (*telno*) is not required. The specified device need not be in the /dev directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* will not give the desired result (see *systemname* below).

-h Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

-t Used to dial an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

-d Causes diagnostic traces to be printed.

-o Designates that odd parity is to be generated for data sent to the remote system.

-n For added security, will prompt the user to provide the telephone number to be dialed rather than taking it from the command line.

-e Designates that even parity is to be generated for data sent to the remote system.

telno When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.

systemname A uucp system name may be used rather than a telephone number; in this case, **cu** will obtain an appropriate direct line or telephone number from /usr/lib/uucp/Systems.

NOTE

The *systemname* option should not be used in conjunction with the **-l** and **-s** options as **cu** will connect to the first available line for the system name specified, ignoring the requested line and speed.

After making the connection, **cu** runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote system so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets these user-initiated commands:

~.	Terminate the conversation.
~!	Escape to an interactive shell on the local system.
~!*cmd...*	Run *cmd* on the local system (via **sh -c**).
~$*cmd...*	Run *cmd* locally and send its output to the remote system.
~%cd	Change the directory on the local system. Note: **~!cd** will cause the command to be run by a sub-shell, probably not what was intended.
~%take *from* [*to*]	Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.

~%put *from* [*to*]	Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places. For both ~%take and put commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
~~ *line*	Send the line ~*line* to the remote system.
~%break	Transmit a BREAK to the remote system (which can also be specified as ~%b).
~%debug	Toggle the -d debugging option on or off (which can also be specified as ~%d).
~t	Print the values of the termio structure variables for the user's terminal (useful for debugging).
~l	Print the values of the termio structure variables for the remote communication line (useful for debugging).
~%nostop	Toggle between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with ~>.

Data from the remote is diverted (or appended, if >> is used) to *file* on the local system. The trailing ~> marks the end of the diversion.

The use of ~%put requires **stty**(C) and **cat**(C) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these cur-

rent control characters on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of **echo**(C) and **cat**(C) on the remote system. Also, *tabs* mode (see **stty**(C)) should be set on the remote system if tabs are to be copied without expansion to spaces.

When **cu** is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. Executing a tilde command reminds the user of the local system uname. For example, uname can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!uname
Y
```

In general, ~ causes the command to be executed on the original machine ~~ causes the command to be executed on the next machine in the chain.

Examples

To dial a system whose telephone number is 9 (201) 555-1212 using 1200 baud (where a dialtone is expected after the 9):

cu -s1200 9=12015551212

If the speed is not specified, "Any" is the default value.

To login to a system connected by a direct line:

cu -l /dev/tty*nn*

or

cu -l tty*nn*

To dial a system with the specific line and a specific speed:

cu -s1200 -l tty*nn*

To dial a system using a specific line associated with an auto dialer:

cu -l cul*nn* **9=12015551212**

To use a system name:

cu *systemname*

Files

/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/spool/locks/LCK..(tty-device)

See Also

cat(C), echo(C), stty(C), uucp(C), uname(C)

Diagnostics

Exit code is zero for normal exit, otherwise, one.

Notes

The **cu** command does not do any integrity checking on data it transfers. Data fields with special **cu** characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a ~. to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **~%put** or **~%take** commands. **Cu** with some modems will not return a login prompt immediately upon connection. A carriage return will return the prompt.

There is an artificial slowing of transmission by **cu** during the **~%put** operation so that loss of data is unlikely.

Cu and **csh**(C) are not compatible.

Name

date - Prints and sets the date.

Syntax

date [**-cms**] [*mmddhhmm[yy]*] [*+format*]

Description

If no argument is given, or if the argument begins with +, the current date and time are printed. If an argument is given, the current date is set. Arguments are

mm= the month number
dd = the day number in the month
hh = the hour number (24-hour clock)
mm= the minute number
yy = the last two digits of the year (optional)

For example,

date 10080045

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is given. The system operates in GMT (Greenwich Mean Time). **Date** takes care of the conversion to and from local standard and daylight time. To change the time zone, see the TZ option of **environ**(C).

To change the time zone,

- Bourne shell (sh)
  Insert the following in /etc/profile:

  TZ=*timezone*
  export TZ

- C shell (csh)
  Insert the following in /etc/cshrc:

  setenv TZ *timezone*

Timezone is either PST, MST, CST, or EST.

If the argument begins with +, the output of **date** is under the control of the user. The format for the output is similar to that of the first argument to **printf**(S). All output fields are of fixed size (zero padded, if necessary). Each field descriptor is preceded by a percent sign (%) and will be replaced in the output by its corresponding value. A single percent sign is encoded by doubling the percent sign, i.e., by specifying "%%". All other characters are copied to the output without change. The string is always terminated with a newline character.

Options

-c Prints the current date and time from the hardware real-time clock; **date -c** *mmddhhmm[yy]* sets the real-time clock.

-m Updates the year on the hardware real-time clock if it is January 1, and makes adjustments to the real-time clock if it is February 29 in a leap year. These dates are not automatically incremented. Be sure to use this option after midnight. The **-m** option checks for January 1 or February 29, and then updates the hardware real-time clock if necessary. For the **-m** option to work correctly, the software clock and the hardware clock should be within twelve hours of one another. Use **cron**(C) to execute date -m each day.

-s Sets (synchronizes) the system (i.e., software) clock to the current time and date from the hardware real-time clock.

The operating system normally uses only the system (software) clock. It uses the hardware real-time clock only with the **date** command.

Field Descriptors:

a	Abbreviated weekday (Sun to Sat)
d	Day of month (01 to 12)
D	Date as mm/dd/yy
h	Abbreviated month (Jan to Dec)
H	Hour (00 to 23)
m	Month of year (01 to 12)
j	Julian date (001 to 366)
M	Minute (00 to 59)
n	Inserts a newline character
r	Time in AM/PM notation
S	Second (00 to 59)
T	Time as HH:MM:SS
t	Inserts a tab character
w	Day of the week (Sunday = 0)
y	Last 2 digits of the year (00 to 99)

Example

If you type

```
date '+DATE:%m/%d/%y%nTIME:%H:%M:%S'
```

the output is:

```
DATE:04/01/85
TIME:14:45:05
```

Related Commands

asktime(C)

Files

/usr/adm/wtmp To record time-setting

Diagnostics

no permission = you aren't the super-user when changing the date.

bad conversion = incorrect syntax used.

bad format character = incorrect field descriptor used.

Name

dc - Desk calculator.

Syntax

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. (See **bc**(C), a preprocessor for **dc** that provides infix notation and a C-like syntax that implements functions. **Bc** also provides reasonable control structures for programs.) 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.

s*x* 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 are printed:

q Exits the program. If executing a string, the recursion level is popped by two.

Q Exits the program. 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 evaluated 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 an operating system command.

c All values on the stack are popped.

i The top value on the stack is poped 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.

;: Used by **bc**(C) for array operations:

Example

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

See Also

bc(C)

Diagnostics

x is unimplemented
Where *x* is an octal number.

stack empty
Not enough elements on the stack to do what was asked.

Out of space
The free list is exhausted (too many digits).

Out of headers
Too many numbers being kept around.

Out of pushdown
Too many items on the stack.

Nesting Depth
Too many levels of nested execution.

Name

dd - Converts and copies a file.

Syntax

dd [*option=value*] ...

Description

The **dd** command 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.

Options	Values
if=*file*	Input file name; standard input is default
of=*file*	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 this 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 (the output file is truncated first)
count=*n*	Copy only *n* input records

conv=ascii	Convert EBCDIC to ASCII
conv=ebcdic	Convert ASCII to EBCDIC
conv=ibm	Slightly different map of ASCII to EBCDIC
conv=lcase	Map alphabetics to lower case
conv=ucase	Map alphabetics to upper case
conv=swab	Swap every pair of bytes
conv=noerror	Do not stop processing on an error
conv=sync	Pad every input record to **ibs**
conv="... , ..."	Several comma-separated conversions

Where sizes are specified, a number of bytes is expected. End a number with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; separate a pair of numbers with **x** to indicate a product.

The **cbs** option is used only if ASCII or EBCDIC conversion is specified. In the former, case characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and newline 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*.

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The **ibm** conversion corresponds better to certain IBM print train conventions.

Newlines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

After completion, **dd** reports the number of whole and partial input and output blocks.

Examples

For example, to read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file x:

dd if=/dev/rct of=x ibs=800 cbs=80 conv=ascii,lcase,sync

Note the use of raw mag tape. **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, type the following:

(dd of=/dev/null; dd of=x) </dev/rct

Related Commands

cp(C), cat(C)

Notes

The **cbs** value must be zero if no block conversion is requested.

The last block of data is not written to tape (/dev/rct) or floppy if there is not enough data to fill that block. Therefore, use the **conv=sync** option to make sure the last block is written.

If you are using raw I/O on the file processor, use a block size that is a mulitiple of 512 bytes.

It's best to use the same block size (used when storing data) when you retrieve data.

Error Messages

f+p records in(out): numbers of full and partial records read(written)

See "Operating System Error Messages" in the *Operations Guide.*

Name

devinfo - Displays device information.

Syntax

/etc/devinfo [**-jp**] [**-b***board*] [**-c***chan*] [**-t***type*]

Description

Use **devinfo** to display information about certain devices in the system. These devices are usually related to communications (e.g., SIO and Multidrop boards). The **devinfo** information comes from the /etc/sysdisp file, if it exists, otherwise from the **sysconf**(S) system call.

The command without any arguments prints information about each board currently in the system that is associated with tty devices. For example,

```
Board  Name       Major  Prefix  First         Last

0      Multidrop  10     tty     /dev/console  /dev/tty63
1      SIO        5      tty     /dev/tty64    /dev/tty73
2      SIO        5      tty     /dev/tty74    /dev/tty83
```

The options are:

-b*board* Displays the type of the specified board as a decimal code. For example, the code for a Multidrop-type board is a 4 and for an SIO-type board is 3.

-c*chan* Displays the tty name associated with channel *chan* on the board specified by the **-b** option.

-j Gives the subjumpering sequence of the board specified by the **-b** option.

-p Displays the entire sysdisp entry.

-t*type* Specifies a sysdisp entry for boards with a type code of *type*. Must be used with the **-p** option.

For example,

devinfo -b1 -c8

displays /dev/tty72, which is the tty name associated with channel 8 on the board in slot 1.

devinfo -b1

displays 3, which is the board type as a decimal code. For more information, see /usr/include/sys/bootinfo.h and the **sysconf** system call.

Files

/etc/sysdisp Describes all possible board types

See Also

sysconf(C), sysconf(S)

Notes

If **devinfo** can't obtain the current board map (the **sysconf**(S) system call was introduced at the same time that Multidrop was added), it assumes that the current system configuration consists of four SIO boards.

As explained in **sysconf**(C), on an Altos 386 Series 1000, an SIO board is reported as a Multidrop. Also, multiple SIOs are reported as only one Multidrop on these systems.

However, the range of tty ports reported in "First" and "Last" is still accurate. For example, a 386 Series 1000 with three SIOs would be reported as shown with only one Multidrop board and 24 ports:

```
Board   Name        Major   Prefix   First   Last
0       multidrop   10      tty      tty00   tty23
```

Name

devnm - Identifies device name on which files reside.

Syntax

/etc/devnm *name...*

Description

Devnm identifies the special file associated with the mounted file system where the argument *name* resides.

Examples

Be sure to type full pathnames as in this example:

/etc/devnm /usr

If /dev/hd0b is mounted on /usr, this produces:

```
hd0b /usr
```

Files

/dev/* device names
/etc/mnttab

Name

diff - Compares two text files.

Syntax

diff [**-befh**] *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 with the name *file2* (*file1*) is used. Normal output line format is:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble ed(C) commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging a for d and reading backward you can ascertain 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 >.

Except in rare circumstances, **diff** finds the smallest sufficient set of file differences. The options are:

-b Causes trailing spaces and tabs to be ignored and other strings of spaces to compare equally.

-e Produces a script of a, c, and d commands for the editor **ed**, which will recreate *file2* from *file1*.

-f Produces a similar script, not useful with **ed**, in the opposite order.

In connection with **-e**, the following shell procedure helps maintain multiple versions of a file:

```
(shift; cat $*; echo '1,$p') | ed - $1
```

which performs a set of editing operations on an original ancestral file. It combines the sequence of **ed** scripts given as all command line arguments except the first. These scripts are presumed to be created with **diff** in the order given on the command line. The set of editing operations is then piped as an editing script to **ed** where all editing operations are performed on the ancestral file given as the first argument on the command line. The final version of the file is then displayed. Only an ancestral file ($1) and a chain of version-to-version **ed** scripts ($2,$3,...) made by **diff** need be on hand.

-h Produces a fast, but less rigorous job. It works only when changed stretches are short and well separated, but it also works on files of unlimited length. The **-e** and **-f** options cannot be used with the **-h** option.

Files

/tmp/d?????
/usr/lib/diffh for -h

See Also

cmp(C), comm(C), ed(C)

Diagnostics

Exit status: 0 = no differences, 1 = differences, 2 = errors.

Missing newline at end of file X. The last line of file *X* did not have a newline. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

Notes

Editing scripts produced under the **-e** or **-f** options do not always work correctly on lines consisting of a single period (.).

Name

diff3 - Compares three files.

Syntax

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 range 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 from 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.

The options are:

-e Publishes a script for the editor, **ed**(C), that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged with "====" and "====3."

-x Produces a script to incorporate changes flagged with "====."

-3 Produces a script to incorporate changes flagged with "====3."

The following command applies an editing script to *file1*:

```
(cat script; echo '1,$p') | ed - file1
```

Files

/tmp/d3*
/usr/lib/diff3prog

See Also

diff(C)

Notes

The **-e** option does not work properly for lines consisting of a single period.

The input file size limit is 64K bytes.

Name

digest - Creates menu system(s) for the Business Shell.

Syntax

digest [*options*] *menufile ...*

Description

Digest is used to create a menu system for use by the Business Shell (**bsh**(C)). This program is also used to modify an existing menu system.

One or more menu systems may be created using the options described below.

-h or **-q** Displays an informative summary of the available options and defaults.

-l *number* Checks for menus longer than *number* lines in length. The default value is 25 if none is specified. This is the correct maximum number for a conventional 24-line CRT screen. In general, *number* should be one larger than the length of the screen area (as defined by "li" in termcap) for the terminal to be used. The user is responsible for ensuring that the width of a menu will fit on the terminal(s) he uses. **Bsh**(C) will truncate lines that are too wide (without issuing a warning message).

-m Multiple menu systems: For each menu file (which must be a directory), this option produces a separate menu system. The names for each menu system are created by suffixing ".bin" to the menu file name.

-o *file* The digested output is sent to the named *file*. By convention, a digested menu system file name should end with a ".bin" suffix.

-s *menu* The starting menu for the generated menu system is the one specified. (This option doesn't make much sense if used with the **-m** option.) If no starting *menu* is specified, the alphabetically first *menu* name is used for each menu system.

-v Verbose: echo menu names as they are processed.

A *menufile* may contain one or more menus or directories containing menus. **Digest** will recursively process all menus within a directory structure.

Note that the **-m** and **-o** options are mutually exclusive. The **-m** option indicates that each menu is to produce a separate ".bin" file: **-o** indicates that a single output file is to be produced with the name given.

The default output file is *menu1*.**bin** if none is specified via the **-o** option, where *menu1* is the first menu file name.

The recommended way to create a menu system is to create a tree of directories containing the various portions of the system. Each subtree contains all the menus related to a given subject. For example, a primary menu (directory) can be created for system management functions and subsidiary menus can be placed beneath (within) the directory for each of the individual system management functions or function areas. Help menus may be placed wherever appropriate in the structure.

Example

Assuming that /usr/lib/menusys contains the following files for the Business Shell menu system:

```
Backup        Execute         Help?     SysAdmin
Backup?       Execute?        Mail      SysAdmin?
Commands?     FloppyBackup    Mail?     TapeBackup
Dir           FloppyBackup?   Start     Tape Backup?
Dir?          Help            Start?
```

Then the following command will make a menu system file:

digest -o /etc/menusys.bin -s Start /usr/lib/menusys

See Also

bsh(C), menus(M), termcap(M)

Diagnostics

The diagnostics produced by **digest** are intended to be self explanatory.

Name

dircmp - Compares directories.

Syntax

dircmp [**-d**] [**-s**] [**-w***n*] *dir1 dir2*

Description

Dircmp examines *dir1* and *dir2* and generates tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated in addition to a list that indicates whether the files common to both directories have the same contents.

The options are:

-d Performs a full **diff**(C) on each pair of like-named files if the contents of the files are not identical.

-s Reports whether the files are "same" or "different."

-w*n* Changes the width of the output line to *n* characters. The default width is 72 characters.

See Also

cmp(C), diff(C)

Name

disable - Disables logins on a port.

Syntax

disable [-d] [-e] tty*nn*...

Description

You must be the super user to use the **disable** command.

The **disable** command manipulates the /etc/inittab file and signals init to disallow logins on a particular port.

Options

-d This option "disables" the port.

-e This option "enables" the port.

Examples

disable tty01

Multiple terminals can be disabled or enabled using the **-d** and **-e** options before the appropriate port:

disable tty01 -e tty02 -d tty03

Files

/etc/inittab

Related Commands

login(M), enable(C), getty(M), ps(C)

Name

dos - Accesses MS-DOS files.

Syntax

doscat [**-r**] *file ...*
dostype [**-r**] *file ...*

doscopy [**-c -r**] *file1 file2*
doscp [**-c -r**] *file1 file2*

doscopy [**-r**] *file ... directory*
doscp [**-r**] *file ... directory*

dosdir *directory ...*

dosls *directory ...*

dosmkdir *directory ...*

dosrm *file ...*
dosdel *file ...*

dosrmdir *directory ...*

Description

The **dos** commands provide access to the files and directories on MS-DOS disks. The commands perform the following actions:

doscat
dostype Copies one or more MS-DOS files to the standard output. If **-r** is given, the files are copied without newline conversions (see the next section, "Conversions.")

doscopy
doscp Copies files between an MS-DOS disk and a UNIX file system. If *file1* and *file2* are given, *file1* is copied to *file2*. If *directory* is given, one or more files are copied to that directory. If **-c** is given, DOS upper-case names are converted to lower-case (for UNIX compatibility). If **-r** is given, the files are copied without newline conversions (see the section titled "Conversions").

dosdir Lists MS-DOS files in the standard MS-DOS style directory format.

dosls Lists MS-DOS directories and files.

dosrm
dosdel Removes files from an MS-DOS disk.

dosmkdir Creates a directory on an MS-DOS disk.

dosrmdir Deletes directories from an MS-DOS disk.

The *file* and *directory* arguments for MS-DOS files and directories have the form:

device:name

where **device** is a UNIX pathname for the special device file containing the MS-DOS disk, and **name** is a pathname to a file or directory on the MS-DOS disk. The two components are separated by a colon (:). For example, the argument:

/dev/fd0:/src/file.asm

specifies the MS-DOS file **file.asm** in the directory **/src** in the device file **/dev/fd0.** Note that slashes, not backslashes, are used as filename separators for MS-DOS pathnames. Arguments without a device name are assumed to be UNIX files.

The **dos** commands operate on the following kinds of floppy disks:

5-1/4 inch MS-DOS
8 or 9 sectors per track
40 tracks per side
1 or 2 sides
MS-DOS version 1, 2, or 3

Conversions

All MS-DOS text files use a carriage return-linefeed combination, CR-LF, to indicate a newline. UNIX uses a single newline LF characters. When the **doscat** and **doscopy** commands transfer MS-DOS text files to UNIX, they automatically remove the CR.

When text files are transferred to MS-DOS, the commands insert the CR before each LF character. You can use the **-r** option to override the automatic conversion and force the command to perform a true byte copy, regardless of file type.

Examples

The following are examples of each type of **dos** command.

To display a file contained on an MS-DOS floppy disk type:

doscat /dev/fd0:/docs/memo.txt

To list the contents of an MS-DOS floppy disk, type one of the following:

dosdir /dev/fd0:

dosls /dev/fd0:

For a Series 500 with Altos System V and MS-DOS partitions, if you are in the active Altos System V partition and want to list files in the DOS partition, enter:

dosls /dev/hd01

where /dev/hd01 is the DOS partition on the hard disk.

On a Series 500, to copy an MS-DOS file from the MS-DOS partition to the current directory in the active Altos System V partition, enter:

doscp /dev/hd01:*filename* .

Or, to copy a file named "notes" from the Altos System V active partition to an existing MS-DOS directory, named "misc," enter:

doscp notes /dev/hd01:/misc/notes

The next command copies a file from an MS-DOS floppy disk to the current directory:

doscopy /dev/fd0:*filename* .

or, for all the files in the root directory of the MS-DOS floppy disk:

doscopy /dev/fd0:"*.*" .

To copy from a UNIX hard disk to an MS-DOS floppy disk, type:

doscopy *filename* **/dev/fd0:**

The next command makes the directory /usr/docs on an MS-DOS disk:

dosmkdir /dev/fd0:/usr/docs

The following command removes the file memo.text from an MS-DOS disk:

dosrm /dev/fd0:/docs/memo.txt

To remove the directory /usr/docs from an MS-DOS disk, type:

dosrmdir /dev/fd0:/usr/docs

See Also

dtype(C)

Files

/dev/fd* Floppy disk devices

Notes

You cannot access MS-DOS directories with wild card specifications. You must make sure you have exclusive access to the device containing the MS-DOS disk. If more than one process tries to access the MS-DOS disk at the same time, the result is unpredictable.

The diskette should be formatted using the **format** command on an MS-DOS system.

Name

drive - Reads drive information written during manufacturing.

Syntax

drive *drive-info option...*

Description

Drive is used by the system during installation. The **drive** command reads the hard disk drive information specified by the *option* from the manufacturing drive information table (*drive-info*) and writes it on the standard output. *Options* include:

cylinders	Number of cylinders on the disk (ST-506 and ESDI drives only).
heads	Number of heads on the disk (ST-506 and ESDI drives only).
spt	Number of sectors per track on the disk (ST-506 and ESDI drives only).
secsize	Number of bytes per sector on the disk (0=512, 1=1024)
skew	Offset between logical sector number zero on one track and logical sector number zero on the next track (default 1).
interleave	Number of sectors between consecutively numbered logical sectors on a track. This number is always 0.
magic	A number indicating that the drive information exists. If the number is 0xd6d1, the drive information exists; otherwise, it does not.
megabytes	Approximate number of megabytes of storage on the disk.

type If a 0 is returned, the drive is an ST-506 drive. If a 1 is returned, the drive is an ESDI drive. If a 2 is returned, the drive is an SCSI drive.

nblocks Number of 512-byte blocks on the disk.

Drive information is recorded on the disk by the company when the computer is manufactured. The following structure shows the format of this information:

```
# pragma pack(2)
struct drive {
   char           dc_jump[3];    /* 3 bytes for a jump instruction */
   char           dc_unused[9];  /* unused */
   unsigned short dc_magic;      /* magic number (0xd6d10 */
   union {
      struct {
                  short dc_cyls; /* number of cylinders */
                  char dc_heads; /* number of heads */
                  char dc_spt;   /* number of sectors per track */
      } dc_hsc;
      unsigned long dc_nblocks;  /* number of sectors for SCSI */
   } dc_un;
   char           dc_secsize;    /* number of bytes per sector */
   char           dc_skew;       /* skew */
   char           dc_interleave  /* interleave */
   char           dc_manutype;   /* code for disk drive manufacturer *

   unsigned short dc_megabytes;  /* approx. number of megabytes */
   unsigned short dc_precmp;
   char           dc_drivetype;
};
#pragma pack()
```

Example

The following command displays the number of cylinders on /dev/hd0:

drive /dev/hd0.drinfo cylinders

Related Commands

layout(C), sizefs(C)

Name

dtype - Determines disk type.

Syntax

dtype [-s] *device ...*

Description

Dtype determines type of disk, prints pertinent information on the standard output unless the silent (-s) option is selected, and exits with a corresponding code (see below). When more than one argument is given, the exit code corresponds to the last argument.

Disk Type	Exit Code	Message (optional)
Misc.	60	error (specified)
	61	empty or unrecognized data
Storage	70	dump format, volume n
	71	tar format[,extent e of n]
	72	cpio format
	73	cpio character (-c) format
MS-DOS	80	MS-DOS 1.x, 8 sec/track, single sided
	81	MS-DOS 1.x, 8 sec/track, dual sided
	90	MS-DOS 8 sec/track, single sided
	91	MS-DOS 8 sec/track, dual sided
	92	MS-DOS 9 sec/track, single sided
	93	MS-DOS 9 sec/track, dual sided
	94	MS-DOS fixed disk
	110	MS-DOS 9 or 15 sec/track, dual sided
UNIX	120	UNIX 2.x filesystem (needs fsck)
	130	UNIX 3.x filesystem (needs fsck)
	131	UNIX 5.x filesystem (or UNIX 3.0 with 1024-byte blocks -- needs fsck)

Notes

UNIX file systems and **dump** and **cpio** binary formats may not be recognized if created on a foreign system. This is due to such system differences as byte and word swapping and structure alignment.

Name

du - Summarizes disk usage.

Syntax

du [**-sar**] [*name ...*]

Description

Du reports the number of blocks contained in all files and directories (recursively) within each directory and file specified by the *name* argument. The block count includes the indirect blocks of the file. If *name* is missing, the current directory is used.

The optional arguments are as follows:

-s Causes the grand total only (for each of the specified *names)* to be given.

-a Causes an output line to be generated for each file.

If neither **-s** or **-a** is specified, an output line is generated for each directory only.

-r Causes **du** to generate messages about directories that cannot be read, files that cannot be opened, etc., rather than being silent (the default).

A file with two or more links is only counted once.

Notes

If the **-a** option is not used, non-directories given as arguments are not listed.

If there are links between files in different directories where the directories are on separate branches of the file system hierarchy, **du** will count the excess files more than once.

Files with holes in them will get an incorrect block count.

Name

dump.hd - Dumps a hard disk to tape.

Syntax

/etc/dump.hd

Description

The **dump.hd** command dumps the entire file system from the hard disk to a cartridge tape. Go to single-user mode (enter **/etc/singleuser**) to guarantee that the hard disk is not being used by any other users while **dump.hd** is running.

Dump.hd only dumps the file system from the first hard disk to tape; it does not dump the second hard disk. If you want to dump the second (third) hard disk to tape, use the **archive**(C) command.

Related Commands

restore.hd(C), archive(C), recover(C)

See Also

Operations Guide

Name

echo - Echoes arguments.

Syntax

/bin/echo [*arg* ...]

Description

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. The arguments are:

-e Prints arguments on the standard output.

-n Prints line without a new line.

-u Uses unbuffered I/O when printing.

-- Prints the arguments exactly so that an argument beginning with a dash (e.g., **-e** or **-n**) can be specified.

Echo also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
****	backslash
\0*n*	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

See Also

sh(C)

Notes

When representing an 8-bit character by using the escape convention **\0***n*, the *n* must **always** be preceded by the digit zero (0).

For example, typing: **echo 'WARNING:\07'** will print the phrase WARNING: and sound the "bell" on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the "\" that precedes the "07". For the octal equivalents of each character, see **ascii**(M).

Name

ed, red - Invokes a line editor.

Syntax

ed [-s] [-p *string*] [*file*]
red [-s] [-p *string*] [*file*]

Description

Ed is the standard line editor. For a full-screen editor, see **vi**(C). If the *file* 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.

-s 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.* Also, see the "Notes" section at the end of this manual page.

-p Allows the user to specify a prompt string to replace the default (*).

Ed operates on a copy of the file it is editing; changes made to 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.* There is only one 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.

Both **ed** and **red** support the **fspec**(F) formatting capability. After including a format specification as the first

line of *file* and invoking **ed** with your terminal in **stty -tabs** or **stty tab3** mode (see **stty**(C)), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed.

NOTE

While inputing text, tab characters when typed are expanded to every eighth column as is the default.

Commands to **ed** have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in 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, followed immediately by a carriage return.

Ed supports a limited form of regular expression notation; regular expressions are used in addresses to specify lines and in some commands (e.g., **s**) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be matched by the RE. The REs allowed by **ed** are constructed as follows:

The following one-character REs match a single character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE 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 RE, or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (dollar sign), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the **g** command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line 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 let-

ters **a** through **f** 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 REs from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches zero or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a range of occurrences of the one-character RE. 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 RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed betweei \(and \) earlier in the same RE. 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 RE 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 RE constrains that RE to match an initial segment of a line.

3.2 A dollar sign ($) at the end of an entire RE constrains that RE to match a final segment of a line.

The construction ^entire RE$ constrains the entire RE to match the entire line. The null RE (e.g.,//) is equivalent to the last RE 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 spepeaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each 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 mark name character *x*, which must be a lower-case letter. Lines are marked with the **k** command described below.

5. A RE enclosed by slashes (/) addresses the first line found by searching forward from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before "Files" below.

6. A RE enclosed in question marks (?) addresses the first line found by searching backward from the line preceding the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. 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 that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) given are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see Rules 5 and 6, above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, 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; they 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 at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)**c**
<text>
.

The change command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)**d**

The delete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

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 file name is given, the currently-remembered file name, if any, is used (see the **f** command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent **e**, **r**, and **w** commands. If *file* is replaced by **!**, the rest of the line is taken to be a shell (**sh**(C)) command whose output is to be read. Such a shell command is not remembered as the current file name. See also "Diagnostics" below.

E *file* The Edit command is like **e**, except that the editor does not check to see if any changes have been made to the buffer since the last **w** command.

f *file* If *file* is given, the file name command changes the currently remembered file name to *file*; otherwise, it prints the currently remembered file name.

(1,$)**g**/*RE*/*command list*
In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; **a**, **i**, and **c** commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the **p** command. The **g**, **G**, **v**, and **V** commands are *not* permitted in the *command list*. See also "Notes" and the last paragraph before "Files" below.

(1,$)**G**/*RE*/
In the interactive **Global** command, the first step is to mark every line that matches the given RE. 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 newline acts as a null command; an & causes the reexecution 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 (Break/Del).

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>
. The insert command inserts the given text before the addressed line; **.** is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the **a** command only in the placement of the input 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 new-line character).

(.,.+1)j The join command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x* The mark command marks the addressed line with name *x*, which must be a lower-case letter. The address **'***x* then addresses this line; **.** is unchanged.

(.,.)l The list command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., tab, backspace) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An **l** command may be appended to any other command other than **e**, **f**, **r**, or **w**.

(.,.)m*a* 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 n command may be appended to any other command other than **e**, **f**, **r**, or **w**.

(.,.)p The print command prints the addressed lines; . is left at the last line printed. The **p** command may be appended to any other command other than **e**, **f**, **r**, or **w**. For example, **dp** deletes the current line and prints the new current line.

P The editor will prompt with the prompt string (* by default) for all subsequent commands. The P command alternately turns this mode on and off; it is initially off unless a prompt string is specified with the **-p** option.

q The quit command causes **ed** to exit. No automatic write of a file is done; however, see "Diagnostics," below.

Q The editor exits without checking if changes have been made in the buffer since the last **w** command.

($)r *file* The read command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see **e** and **f** commands). The currently-remembered file name is not changed unless *file* is the very first file name mentioned since **ed** was invoked: Address 0 is legal for **r** and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by **!**, the rest of the line is taken to be a shell (**sh**(C)) command whose output is to be read: For example, **"$r !ls"** appends a listing of the current directory to the end of the file being edited. Such a shell command is not remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/g or
(.,.)s/*RE*/*replacement*/n **n = 1-512**

The substitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* 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. If a number *n* appears after the command, only the *n*th occurrence of the matched string on each addressed line 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 RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before "Files" below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character **%** is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The **%** loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it with a \. Such substitution cannot be done as part of a **g** or **v** command list:

(.,.)t*a* 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 at 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/*RE*/*command list*
This command is the same as the global command **g** except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE*/
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 RE.

(1,$)w *file*
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 **umask**(C)) dictates otherwise. The currently-remembered file name is not changed unless *file* is the very first file name mentioned since **ed** was invoked. If no file name is given, the currently-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 typed. If *file* is replaced by **!**, the rest of the line is taken to be a shell (**sh**(C)) command whose standard input is the addressed lines. Such a shell command is not remembered as the current file name.

X An encryption key is requested from the standard input. Subsequent **e**, **r**, and **w** commands will use this key to encrypt or decrypt the text. An explicitly empty key turns off encryption. Also, see the **-x** option of **ed**.

(**$**)= 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 operating system shell (**sh**(C)) 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**)<new-line>
An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (Break/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, and 64 characters per file name, the limit on the number of lines depends on the amount of user memory.

When reading a file, **ed** discards ASCII NUL characters. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by **ed**.

If a file is not terminated by a new-line character, **ed** adds one and outputs a message explaining what it did.

If the closing delimiter of a RE 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

/usr/tmp	default directory for temporary work file
$TMPDIR	if this environmental variable is not null, its value is used in place of /usr/tmp as the directory name for the temporary work file
ed.hup	work is saved here if the terminal is hung up

Diagnostics

?	For command errors.
?*file*	For an inaccessible file. (use the **help** and **Help** commands for detailed explanations).

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, **ed** warns the user if an attempt is made to destroy **ed**'s buffer via the **e** or **q** commands. It prints **?** and allows one to continue editing. A second **e** or **q** command at this point will take effect. The **-s** command-line option inhibits this feature.

See Also

edit(C), ex(C), grep(C), sed(C), sh(C), stty(C), umask(C), vi(C) fspec(F), regexp(S) in the *Reference (CP, S, F)*

Notes

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 editor is invoked from a restricted shell (see **sh**(C)). The sequence **\n** in a RE does not match a new-line character. Characters are masked to 7 bits on input. If the editor input is coming from a command file (e.g., **ed file < ed-cmd-file**), the editor will exit at the first failure.

The - option, although supported in this release for upward compatibility, will no longer be supported in the next major release of the system. Convert shell scripts that use the - option to use the -s option, instead.

Name

edit - Text editor (variant of ex for casual users).

Syntax

edit [**-r**] *name...*

Description

Edit is a variant of the text editor **ex** recommended for new or casual users who wish to use a command-oriented editor.

-r Recover file after an editor or system crash.

This brief introduction should help you get started with **edit.** To edit the contents of an existing file, enter **edit** *filename.* **Edit** makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run **edit** on it.

Edit prompts for commands with a colon (:), which you should see after starting the editor. If you are editing an existing file, then you will have some lines in **edit**'s buffer (its name for the copy of the file you are editing). Most commands to **edit** use its "current line" if you do not tell them which line to use. Thus, if you say **print** (which can be abbreviated **p**) and press Retn (as you should after all **edit** commands), this current line will be printed. If you **delete** (**d**) the current line, **edit** will print the new current line. When you start editing, **edit** makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**), **edit** will read lines from your terminal until you give a line consisting of just a period (.), placing these lines after the current line. The last line you type then

becomes the current line. The command **insert** (**i**) is like append but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command **1** then **edit** will type this first line. If you then give the command **delete**, **edit** will delete the first line, line 2 will become line 1, and **edit** will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You type **s/***old*/*new*/ where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file, you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run **edit** on a file, but do not change it, it is not necessary (but does no harm) to write the file back. If you try to quit from **edit** after modifying the buffer without writing it out, you will be warned that there has been "No write since last change" and **edit** will wait for another command. If you don't want to write the buffer out, then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file, you can make any changes you desire. You should learn at least a few more things, however, if you are to use **edit** more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append**) you give lines up to a line consisting of only a period (.). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., **3,5change**. You can print lines this way too. Thus **1,23p** prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command taht does not do what you want, you can use the **undo** command to restore the old contents of the line. You can also undo an **undo** command.

Edit will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an **undo** and looking to see what happened. If you decide that the change is ok, then you can press **u** again to get it back. Note that commands such as **write** and **quit** cannot be undone.

To look at the next line in the buffer, just press Retn. To look at a number of lines, press Ctrl-d (press the Ctrl key and, while it is held down, press the **d** key) rather than Retn. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around you by giving the command, **z**. The current line will then be the last line printed; you can get back to the line where you were before the **z** command by typing a double quote ("**)**.

The **z** command can also be given other characters: **z+** prints a screen of text (or 24 lines) ending where you are; **z+** prints the next screenful. If you want less than a screenful of lines, type in **z.12** to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command **delete 5.**

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form */text/* to search forward for *text*, or *?text?* to search backward for *text*. If a search reaches the end of the file without finding the text, it goes back to the beginning, and continues to search back to the line where you are. A useful feature here is a search of the form */^text/* which searches for *text* at the beginning of a line. Similarly, */text$/* searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name dot (.); this is most useful in a range of lines as in **.,$print** which prints the rest of the lines in the file. To get to the last line in the file, you can refer to it by its symbolic name "$". Thus the command **$ delete** or **$d** deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$-5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out the current line by typing .=. This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move, you can then enter **10,20delete a,** which deletes these lines from the file and places them in a buffer named **a.** **Edit** has 26 such buffers named **a** through **z.** You can later get these lines back by doing "put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files, you can give an **edit** (e) command after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing **delete** to **yank** above, you can get a pattern for copying lines. If the text you wish to move or copy is all within one file, then you can just type **10,20move $,** for example. It is not necessary to use named buffers in this case (but you can if you wish).

See Also

id(C), ex(C), vi(C)

Name

egrep - Searches a file for a pattern using full regular expressions.

Syntax

egrep [*options*] *full regular expression* [*file ...*]

Description

Egrep (*expression grep*) searches files for a pattern of characters and prints all lines that contain that pattern. **Egrep** uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

Egrep accepts full regular expressions as in **ed**(C), except for \(and \), with the addition of:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.

2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.

3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.

4. A full regular expression that may be enclosed in parentheses () for grouping.

Be careful using the characters **$**, *, [, ^, |, (,), and \ in *full regular expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes '...'.

The order of precedence of operators is [], then *?+, then concatenation, then | and new-line.

If no files are specified, **egrep** assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-h Suppress the filename header at the beginning of each line.

-i Ignore upper/lower case distinction during comparisons.

-l Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-s Suppress the error message for an inaccessible file.

-v Print all lines except those that contain the pattern.

-e *special_expression*
Search for a *special expression* (*full regular expression* that begins with a -).

-f *file*
Take the list of *full regular expressions* from *file.*

See Also

ed(C), fgrep(C), grep(C), sed(C), sh(C)

Diagnostics

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

Notes

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in /usr/include/stdio.h.

Name

enable - Enables logins on a port.

Syntax

enable [-d] [-e] tty*nn*...

Description

The **enable** command manipulates the /etc/inittab file and signals **init**(M) to allow logins on a particular port. (However, never enable a printer port.) You must be the super-user to use this command.

Options

-d,-e The **-d** and **-e** options can be used in a single command line to enable (**-e**) login on some ports and disallow (**-d**) logins on others.

Examples

In the example below, tty01 is enabled:

enable tty01

In the next example, the **-d** and **-e** options are put in before the appropriate port names to allow and disallow logins on those ports.

enable console -e tty02 -d tty03 tty04

Related Commands

login(M), disable(C), inittab(M)

Files

/etc/inittab

Name

enroll, xsend, xget - Secret mail.

Syntax

enroll
xsend *person*
xget

Description

These commands implement a secure communication channel; like **mail**(C), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use **enroll;** it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use **xget.** It asks for your password, then gives you the messages. Typing a **?** displays a menu of valid **xget** commands.

To send secret mail, use **xsend** in the same manner as the ordinary mail command. (However, it will accept only one target.) A message announcing the receipt of secret mail is also sent by ordinary mail.

Files

/usr/spool/secretmail/*.key	Public keys
/usr/spool/secretmail/*.[0-9]	Messages

See Also

mail(C)

Name

env - Sets environment for command execution.

Syntax

env [-] [*name=value ...*] [*command args*]

Description

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The - flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments. If no command is specified, the resulting environment is printed, one name/value pair per line.

See Also

sh(C), environ(M), profile(M), and exec(S) in the *Reference (CP, S, F)*

Name

errstop - Terminates the error-logging daemon.

Syntax

/etc/errstop

Description

The error-logging daemon **strerr**(M) is terminated by using **errstop.** This is accomplished by executing **ps**(C) to determine the daemon's identity and then sending it a software kill signal. Only the super-user may use **errstop.**

See Also

ps(C), kill(C), strerr(M) and signal(S) in the *Reference (CP, S, F)*

Name

ex - Invokes a text editor.

Syntax

ex [-] [**-v**] [**-t** *tag*] [**-r** *file*] [**-L**] [**-R**]
[**-c** *command*] *file* ...

Description

Ex is the root of a family of editors: **ex** and **vi.** Ex is a superset of **ed**(C), with the most notable extension being a display editing facility. Display-based editing is the focus of **vi**(C).

For ed Users

If you have used **ed** you will find that **ex** has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with **vi.** Generally, the editor uses far more of the capabilities of terminals than **ed** does, and uses the terminal capability data base and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor uses features such as insert and delete character and line in its **visual** command (abbreviated **vi**); this is the central mode of editing when using **vi.**

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Pressing Ctrl-d causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just pressing Retn. Of course, the screen-oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command lets you reverse any single change that goes astray. Ex gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command. So it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a **write** a file other than the one you are editing. If the system (or editor) crashes, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '**%**' is also available in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named **a** through **z**. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in **ex** which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. **Ex** also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use Ctrl-d as a backtab and space and tab forward to alig new code easily.

Miscellaneous new useful features include an intelligent **join** (**j**) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as **sort**.

Invocation Options

The following invocation options are interpreted by **ex:**

-	Suppresses all interactive-user feedback. This is useful in processing editor scripts.
-v	Invokes **vi.**
-t *tag*	Edits the file containing the *tag* and position the editor at its definition.
-r *file*	Recovers *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
-L	Lists the names of all files saved as the result of an editor or system crash.
-R	Sets **readonly** mode, which prevents accidentally overwriting the file.
-c *command*	Begins editing by executing the specified editor search or positioning *command.*
file	Indicates files to be edited.

Ex States

Command	Normal and initial state. Input prompted for by **:.** Your kill character cancels partial command.
Insert	Entered by **a, i,** or **c.** Arbitrary text may be entered. Insert is normally terminated by a line having only . on it, or abnormally with an interrupt.
Visual	Entered by **vi,** terminates with **Q** or ^\.

Ex Command Names and Abbreviations

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**	.	.	unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	window	**z**
insert	**i**	rewind	**rew**	escape	**!**
join	**j**	set	**se**	lshift	**<**
list	**l**	shell	**sh**	print next	CR
map		source	**so**	resubst	**&**
mark	**ma**	stop	**st**	rshift	**>**
move	**m**	substitute	**s**	scroll	**^D**

Ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
-	previous	*'x*	marked with *x*
+n	*n* forward	"	previous context
%	1,$		

Initializing Options

EXINIT	Place **set**s here in environment var
$HOME/.exrc	Editor initialization file
./.exrc	Editor initialization file
set *x*	Enable option
set no*x*	Disable option
set *x=val*	Give value *val* to option *x*
set	Show changed options
set all	Show all options
set *x***?**	Show value of option *x*

Most Useful Options

autoindent	ai	Supply indent
autowrite	aw	Write before changing files
ignorecase	ic	In scanning
list		Print ^I for tab, $ at end
magic		.[* special in patterns
modelines		First five lines and last five lines executed as **vi/ex** commands if they are in the form **vi:***command*: or **ex:***command*:
number	nu	Number lines
paragraphs	para	Macro names which start...
redraw		Simulate smart terminal
report		Informs you if the number of lines modified by the last command is greater than the value of the **report** variable command mode lines
scroll		Command mode lines
sections	sect	Macro names...
shiftwidth	sw	For <>, and input ^D
showmatch	sm	To) and } as typed
showmode	smd	Show insert mode in **vi**
slowopen	slow	Stop updates during insert
term		Specifies to **vi** the type of terminal being used (the default is the term value of the environmental variable TERM)
window		Visual mode lines
wrapscan	ws	Around end of buffer?
wrapmargin	wm	Automatic line splitting

Scanning Pattern Formation

^	Beginning of line
$	End of line
.	Any character
\<	Beginning of word
\>	End of word
[*str*]	Any char in *str*
[^*str*]	Not in *str*
[*x*-*y*]	Between *x* and *y*
*	Any number of preceding

Author

Vi and ex are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

Files

/usr/lib/ex.strings	Error messages
/usr/lib/ex.recover	Recover command
/usr/lib/ex.preserve	Preserve command
/usr/lib/*/*	Describes capabilities of terminals
$HOME/.exrc	Editor startup file
./.exrc	Editor startup file
/tmp/Ex*nnnnn*	Editor temporary
/tmp/Rx*nnnnn*	Named buffer temporary
/usr/preserve/*login*	Preservation directory (where *login* is the user's login)

See Also

awk(C), ed(C), edit(C), grep(C), sed(C), vi(C) term(M), terminfo(M) and curses(S) in the *Reference (CP, S, F)*

Notes

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 do not print a name if the command line '-' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor. Null characters are discarded in input files and cannot appear in resultant files.

Name

expr - Evaluates arguments as an expression.

Syntax

expr *arguments*

Description

The *arguments* are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that zero is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within braces ({ and }).

expr \| *expr*

Returns the first *expr* if it is neither null nor 0, otherwise returns the second *expr.*

expr \& *expr*

Returns the first *expr* if neither *expr* is null nor 0, otherwise returns 0.

expr { =, \>, \>=, \<, \<=, != } *expr*

Returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, - } *expr*

Addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*

Multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*

The matching operator : compares the first argument with the second argument which must be a regular expression; regular expression syntax is the same as that of **ed**(C), except that all patterns are "anchored" (i.e., begin with a caret (^)) and therefore the caret is not a special character in that context. (Note that in the shell, the caret has the same meaning as the pipe symbol (|).) Normally the matching operator returns the number of characters matched (zero on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

Examples

To add 1 to the shell variable *a*:

```
a=`expr $a + 1`
```

For $a equal to either "/usr/abc/file" or just "file,"

```
expr $a : '.*/\(.*\)' \| $a
```

returns the last segment of the pathname (i.e., file). Watch out for the slash alone as an argument; **expr** will take it as the division operator (see "Notes" below).

Even better and more simple than the above expression, add the // characters to eliminate any ambiguity about the division operator:

```
expr //$a : '.*/\(.*\)'
```

To return the number of characters in $VAR:

```
expr $VAR : '.*'
```

Related Commands

ed(C), sh(C)

Diagnostics

As a side effect of expression evaluation, **expr** returns the following exit values:

0 If the expression is neither null nor zero

1 If the expression is null or zero

2 For invalid expressions

Other diagnostics include:

syntax error	For operator/operand errors
nonnumeric argument	If arithmetic is attempted on such a string

Notes

After argument processing by the shell, **expr** cannot tell the difference between an operator and an operand except by the value. If $a is an equal sign (=), the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

Thus the arguments are passed to **expr** (and will all be taken as the = operator). The following permits comparing equal signs:

```
expr X$a = X=
```

Name

factor - Factors a number.

Syntax

factor [*integer*]

Description

When **factor** is invoked without an argument, it waits for a *number* to be typed. If you type in a positive number less than 10^{14}, 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*. **Factor** exits if it encounters a zero or any nonnumeric character.

If **factor** is invoked with an argument, it factors the number as above and then exits.

The time it takes to **factor** a number, *n*, is proportional to sqrt(*n*). It usually takes longer to factor a prime or the square of a prime, than to factor other numbers.

Diagnostics

Factor returns an error message if the supplied input value is greater than 10^{14} or if it is not an integer number.

Name

false - Returns with a nonzero exit value.

Syntax

false

Description

The **false** command returns a nonzero exit value and is typically used in shell procedures.

Example

Following is an example of using false in a shell procedure:

```
until false
do
     command
done
```

Related Commands

sh(C), true(C)

Diagnostics

False has exit status 1.

Name

fcopy - Copies a floppy diskette.

Syntax

fcopy

Description

Use the **fcopy** command to make duplicate copies of a floppy diskette. The routine is menu driven and will prompt you when to insert and remove the diskette. After one copy has been made, you can make additional copies of the same diskette.

All new diskettes must be formatted before they can be copied (see **format**(C)).

Example

To copy a floppy diskette, type:

fcopy

and press Retn. The screen displays the following:

```
1 - Copy low density floppy diskette
2 - Copy high density floppy diskette
3 - Quit
```

Select the option you want, insert the diskette you want to copy, and press Retn. The system copies a certain amount of data from the diskette to a temporary file on the hard disk. Then you are prompted to:

```
Insert blank diskette, press RETURN
```

During this phase, the system copies the data from the hard disk to the blank diskette.

Depending on the amount of data, you may be prompted to repeat the above procedure.

Files

/tmp/junk.??????	Temporary working file, created and subsequently removed by **fcopy.**

Related Commands

format(C), dd(C)

Name

fdisk - Maintains disk partitions (Series 500 only).

Syntax

fdisk [[**-p**] [**-n**] [**-x**] [**-ad** *partition*] [**-c** *start size type*] [**-f** *devicename*]]

Description

Fdisk displays information about disk partitions. **Fdisk** also creates and deletes disk partitions and changes the active partition. **Fdisk** functionality is a superset of the MS-DOS command of the same name. **Fdisk** is usually used interactively from a menu.

The hard disk has at most four partitions. Only one partition is active at any given time. It is possible to assign a different operating system to each partition. Once a partition is made active, the operating system resident in the partition boots automatically once the current operating system is halted.

To use Altos System V, at least one partition must be assigned to Altos System V.

The "Use Entire Disk for Altos System V" option always leaves the first track unassigned. The first track on the hard disk is reserved for masterboot.

For example, if a disk has 2442 tracks, **fdisk** reports these as tracks 0-2441. **Fdisk** will assign (using the "Use Entire Disk for Altos System V" option) tracks 1-2441. (Track 0 is reserved for masterboot.)

Partitions are defined by a "partition table" at the end of the master boot block. The partition table provides the location and size of the partitions on the disk. The partition table also defines the active partition. Each partition can be assigned to Altos System V, DOS or some other operating system. The DOS partition must be formatted using the DOS **format** command. Once a DOS partition is set up, DOS **files** and directories resident in the DOS partition may then be accessed while running Altos System V by means of the **dos**(C) commands.

Arguments

These flags are used to invoke **fdisk** non-interactively:

-a *number* — Activates the specified partition number.

-c *start size type* — Creates partition with specified *start*, *size*, and *type*; *start* and *size* are specified in tracks, and *type* is one of:

1 = Altos System V partition
2 = DOS partition

-d *number* — Deletes the specified partition number.

-f *name* — Opens device *name* and reads the partition table associated with the device's partition. The default is **/dev/rhd0.entire.**

-n — Deletes all partitions and removes the masterboot. The disk must be completely re-installed.

-p — Prints out the partition table. Displays the partition number, start, end, size, and type; start, end, and size are given in tracks.

-x — Uses the entire disk for UNIX.

Options

The **fdisk** command displays a prompt and a menu of options. Updates to the disk are not made until you enter "q" from the main menu.

1. Display Partition Table.

   This option displays a table of information about each partition on the hard disk. The PARTITION column gives the partition number. The STATUS column tells whether the partition is active (A) or inactive (I). TYPE tells whether the partition is Altos System V, DOS, or "other." the option also displays the starting track, ending track and total number of tracks in each partition.

2. Use Entire Disk for Altos System V

   Fdisk creates one partition that includes all the tracks on the disk, except the first track and the last cylinder. This partition is assigned to Altos System V and is designated the active partition.

3. Create a Partition

   The option allows the creation of a partition by altering the partition table. **Fdisk** reports the number of tracks available for each partition and the number of tracks in use. **Fdisk** prompts for the partition to create, the starting track, size in tracks, and partition type. The change is written to the operating system and the hard disk when you enter "q" from the main menu.

4. Activate Partition

   This option activates the specified partition. Only one partition may be active at a time. The change is not effective until you exit. The operating system residing in the newly activated partition boots once the current operating system is halted.

5. Delete Partition

   This option requests which partition you wish to delete. **Fdisk** reports the new available amount of disk space in tracks. The change is not effective until you exit.

   Exit the **fdisk** program by typing a "q" at the main **fdisk** menu. Your changes are now written to the operating system and the hard disk.

Notes

The minimum recommended size for an Altos System V partition on the first hard disk is 20 megabytes.

Since **fdisk** is intended for use with DOS, it may not work with all operating system combinations.

Name

fgrep - Searches a file for a character string.

Syntax

fgrep [*options*] *string* [*file...*]

Description

Fgrep (fast **grep**) searches files for a character string and prints all lines that contain that string. **Fgrep** is different from **grep**(C) and **egrep**(C) because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters **$**, *, [, ^, |, (,), and \ are interpreted literally by **fgrep**; that is, **fgrep** does not recognize full regular expressions as does **egrep**. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes '...'.

If no files are specified, **fgrep** assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-h Suppress the filename header at the beginning of each line.

-i Ignore upper/lower case distinction during comparisons.

-l Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-s Suppress the error message for an inaccessible file.

-v Print all lines except those that contain the pattern.

-x Print only lines matched entirely.

-y Same as **-i** option.

-e *special_string*
Search for a *special string* (*string* begins with a -).

-f *file*
Take the list of *strings* from *file*.

See Also

ed(C), egrep(C), grep(C), sed(C), sh(C)

Diagnostics

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

Notes

Ideally there should be only one **grep** command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in /usr/include/stdio.h.

Name

file - Determines file type.

Syntax

file [**-c**] [**-f** *ffile*] [**-m** *mfile*] *arg...*

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. If an argument is an executable **a.out**, **file** will print the version stamp, provided it is greater than 0.

-c The **-c** option causes **file** to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **-c**.

-f If the **-f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

-m The **-m** option instructs **file** to use an alternate magic file.

File uses the file /etc/magic to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of /etc/magic explains its format.

Files

/etc/magic

See Also

filehdr(F) in the *Reference (CP, S, F)*

Name

find - Finds files that match certain conditions.

Syntax

find *pathname-list expression*

Description

The **find** program recursively searches the directory hierarchy for each path name in the *pathname-list*, looking for 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*.

Options

-atime *n* — True if the file has been accessed in *n* days. The access time of directories in *pathname-list* is changed by **find** itself.

-cpio *device* — Always true; write the current file on *device* in **cpio**(C) format (5120-byte records).

-ctime *n* — True if the file status (mode, ownership, links) has been changed in *n* days.

-depth — Always true; causes descent of the directory hierarchy so that all entries in a directory are acted on before the directory itself. This can be useful when **find** is used with **cpio**(C) to transfer files that are contained in directories without write permission.

-exec *cmd* — True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by a space and an escaped semicolon. A command argument {} is replaced by the current pathname.

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

-group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the /etc/group file, it is taken as a group ID.

-inum *n* True if the file has the specified inode number, *n*.

-links *n* True if the file has *n* links.

-local True if the file physically resides on the local system.

-mount Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.

-mtime *n* True if the file data has been modified in *n* days.

-name *file* True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for the left bracket ([), the question mark (?) and the star (*)).

-newer *file* True if the current file has been modified more recently than the argument file.

-nosym Does not descend into directories that are symbolic links.

-ok *cmd* Like **-exec** except that the generated command line is displayed with a question mark first, and is and is executed only if the user responds by typing y.

-perm *onum* — True if the file permission flags exactly match the octal number *onum* (see **chmod**(C)). If *onum* is prefixed by a minus sign, more flag bits (017777, see **stat**(S)) become significant and the flags are compared:

(flags&onum)==onum

-print — Always true; causes the current pathname to be printed.

-size *n*[**c**] — True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.

-type *c* — True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **l**, **p**, **or f**, for block special file, character special file, directory, symbolic link, named pipe, or plain file.

-user *uname* — True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the /etc/passwd file, it is taken as a user ID.

The primaries may be combined using the following operators (in order of decreasing precedence):

negation — The negation of a primary is specified with the exclamation (!) unary NOT operator.

AND — The AND operation is implied by the juxtaposition of two primaries.

OR — The OR operation is specified with the **-o** operator given between two primaries.

Examples

In the example below all the files named a.out or *.o that have not been accessed for a week are found and removed.

```
find / \( -name a.out -o -name '*.o' \) \( -atime +7 \)
-exec rm {} \;
```

See Also

chmod(C), cpio(C), sh(C), test(C), stat(S), umask(S)

Files

/etc/passwd
/etc/group

Name

finger - Finds information about users.

Syntax

finger [**-bfilpqsw**] [*login* ...]

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, office location, and phone number (if known) for each current user. (Idle time is in minutes if it is a single integer, hours and minutes if a colon (:) is used, or days and hours if a "d" is used.)

A longer format also exists and is used by **finger** whenever a list of names is given. (Account names as well as first and last names of users are accepted.) This is a multi-line format; including all of the information described above as well as the user's home directory and login shell, any plan which the person has placed in the .plan file in their home directory, and the project on which that user is working from the .project file, also in the home directory.

Options are:

-b Prints briefer long output format of users.

-f Suppresses the printing of the header line (short format).

-i Prints quick list of users with idle times.

-l Forces long output format.

-p Suppresses printing of the .plan files.

-q Prints quick list of users.

-s Forces short output format.

-w Forces narrow format list of specified users.

Files

/etc/utmp	Who file
/etc/passwd	User names, offices, phones, login directories, and shells
$HOME/.plan	Plans
$HOME/.project	Projects

See Also

who(C)

Notes

Only the first line of the .project file is printed.

The "office" column of the output will contain any text in the comment field of the user's /etc/passwd file entry that immediately follows a comma (,). For example, if the entry is:

johnd:eX8HinAk:201:50:John Doe, 321:/usr/johnd:/bin/sh

the number 321 will appear in the office column.

Idle time is computed as the elapsed time since any activity on the given terminal. This includes previous invocations of **finger** which may have modified the terminal's corresponding device file, /dev/tty??.

This utility was developed at the University of California at Berkeley and is used with permission.

Name

fleece - Looks for files in home directories.

Syntax

fleece *file*

Description

Fleece looks for the named *file* in every home directory on the system and lists those which exist.

Files

/etc/passwd	To find home directories

Name

fmt - Simple text formatter.

Syntax

fmt [*file...*]

Description

Fmt is a simple text formatter that reads the concatenation of input files (or standard input if none are given) and produces on the standard output a version of its input with lines as close to 72 characters long as possible. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

Fmt is meant to format mail messages prior to sending, but may also be useful for other simple tasks. For instance, within **vi**(C), the command:

!}fmt

will reformat a paragraph, evening the lines.

See Also

mail(C), nroff(1)

Notes

The program was designed to be simple and fast; for more complex operations, use the standard text processors.

Name

fold - Folds long lines for finite width output device.

Syntax

fold [*-width*] [*file ...*]

Description

Fold is a filter that will fold the contents of a specified file, breaking the lines to fit a maximum width. If no file name is given, the program will use the standard input.

Fold accepts the following option:

-width The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using **expand**(C) before using the **fold** command.

Name

format - Formats a floppy diskette.

Syntax

format

Description

Format is a menu-driven program for formatting floppy disks. Disks are formatted in a 5-1/4 inch, double-density, double-sided format.

For Altos systems with a dual-speed floppy drive, when you type **format**, the screen looks like this:

```
1 - Altos format                /dev/fd096ds9   (96 tpi 9 sec/trk)
2 - IBM-AT (slow) format XENIX /dev/fd048ds9   (48 tpi 9 sec/trk)
3 - IBM-AT (fast) format XENIX /dev/fd096ds15  (96 tpi 15 sec/trk)
4 - Quit

Command: [default Altos]
```

Type **1**, then Retn and you are prompted to insert a blank diskette and press Retn. A series of dots (.....) will appear on the screen. When the diskette is formatted the format menu reappears. Type **4** to quit; the system prompt returns to the screen.

CAUTION

The computer has a dual-speed floppy disk drive. Floppy disks designed for a high speed drive cannot be used on a low speed drive, and floppy disks designed for a low speed drive cannot be used on a high speed drive.

The system will determine what type of floppy disk you have before it begins copying files to the floppy disk.

Files

/usr/lib/ffmt

See Also

fcopy(C), dd(C)

Name

from - Who is my mail from?

Syntax

from

Description

From lists the mail header lines in your mailbox file, to show you who your mail is from.

Files

/usr/spool/mail/*

Name

fsck, dfsck - Checks and repairs file systems.

Syntax

/etc/fsck [*options*] [*file-system*]
/etc/dfsck [*options1*] *fsys1 ...* - [*options2*] *fsys2 ...*

Description of fsck

The **fsck** command must be run on the root device in single-user mode.

The **fsck** command performs a file system check by auditing and interactively repairing inconsistencies in the file system. 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, you are prompted for agreement before each correction is attempted. The system waits for you to respond yes or no. Most corrective actions result in some loss of data. The amount and severity of the loss may be determined from the diagnostic output. If you do not have write permission, **fsck** defaults to the action of the **-n** option.

Inconsistencies checked are as follows:

- Blocks claimed by more than one inode or the free list
- Blocks claimed by an inode or the free list outside the range of the file system
- Incorrect link counts
- Size checks:
  Incorrect number of blocks
  Directory size not 16-byte aligned
- Bad inode format
- Blocks not accounted for anywhere
- Directory checks:

File pointing to unallocated inode
Inode number out of range

- Super-block checks:
  More than 65536 inodes
  More blocks for inodes than there are in the file system

- Bad free block list format

- Total free block or free inode count incorrect

Orphaned files and directories (allocated but unreferenced) are 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 pre-exist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished (when the system is installed) by making lost+found, copying a number of files to the directory, and then removing them before **fsck** is executed.

Options

-b Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.

-D Checks directories for bad blocks (useful after system crash).

-f Does a fast check of the *file system* (blocks and sizes and free list checks). Reconstructs free list if necessary.

-h Complains about files whose byte and block counts don't match.

-n Assumes a "no" response to all questions asked by **fsck**; does not open the file system for writing.

-q Quiet **fsck**. Assumes yes in response to most questions. Unreferenced fifos will be silently removed. If required, counts in the superblock will be corrected.

-sX Ignores the actual free list and (unconditionally) reconstructs a new one by rewriting the super-block of the file system. The file system must be unmounted while this is done. If this is not possible, 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 -sX option allows for creating an optimal freelist organization. The format for X is *cylinder size:gap size*. If X is not given, then the values used when the file system was created are used.

-SX Conditionally reconstructs the free list. This option is like -sX except that the free list is rebuilt only if there are no discrepancies discovered in the file system. Using -S forces a "no" response to all questions asked by **fsck**. This option is useful for forcing free list reorganization on uncontaminated file systems.

-t If **fsck** cannot obtain enough memory to keep its tables, it uses a scratch file. If the -t option is specified, the file named in the next argument is used as the scratch file, if needed. Without -t, **fsck** prompts for the name of the scratch file. The file chosen 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.

-y Assumes a "yes" response to all questions asked by **fsck**.

If no file systems are specified, **fsck** reads a list of default file systems from the file /etc/checklist.

Description of dfsck

Dfsck allows two file system checks on two different drives simultaneously. *Options1* and *options2* are used to pass options to fsck for the two sets of file systems. A dash (-) is the separator between the file system groups.

Dfsck permits you to interact with two **fsck** programs at once. To aid this, **dfsck** will print the file system name for each message.

When answering a question from **dfsck**, you must prefix the response with a **1** or a **2** to indicate that the answer refers to the first or second file system.

Do not use **dfsck** to check the root file system.

Examples

For example, to check the main hard disk, type:

```
fsck /dev/root
```

For the second hard disk, the procedure is as follows. If the second hard disk is mounted, type:

```
/etc/umount /dev/hd1b
fsck /dev/hd1b
```

To remount the second hard disk back to usr2, type:

```
/etc/mount /dev/hd1b /usr2
```

If the file system is in good order, the screen displays:

```
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Count
** Phase 5 - Check Free List
```

If the second hard disk is not mounted, skip the umount and mount steps. If you have a third hard disk, substitute **hd2b** for **hd1b** and **/usr3** for **/usr2.**

The following example shows file system inconsistencies:

```
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Count
UNREF FILE I = 2124  OWNER=CHRIS MODE=100644
SIZE=30574  MTIME=Apr 27 07:56 1983
CLEAR?  y

** Phase 5 - Check Free List
63 BLK(S) MISSING
BAD FREE LIST
SALVAGE?  y
```

The system automatically clears and salvages the file system, and the following message apppears:

```
Normal System Shutdown.
```

The system should automatically reboot after **fsck** shuts it down.

Files

/etc/checklist	Contains default list of file systems to check

See Also

mkfs(M), ncheck(M), checklist(M), filesystem(M)

Notes

Inode numbers for . and .. in each directory are not checked for validity.

The **fsck** program will not run on a mounted non-raw file system unless the file system is the root file system or unless the **-n** option is specified and no writing out of the file system will take place. If any such attempt is made, a warning is displayed on the screen and no further processing of the file system is done for the specified device.

Checking the raw device is almost always faster and should be used with everything but the root file system.

Although checking a raw device is almost always faster, there is no way to tell if the file system is mounted. And cleaning a mounted file system will almost certainly result in an inconsistent superblock.

Unreferenced files of size 0 are removed without asking first.

Name

getopt - Parses command options.

Syntax

set -- `getopt** *optstring* **$*`

Description

Getopt is used to check and break up options in command lines for parsing by shell procedures. *Optstring* is a string of recognized option letters (see **getopt**(S)). If a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by whitespace. The special option -- is used to delimit the end of the options. **Getopt** will place -- in the arguments at the end of the options, or recognize it if it is used explicitly. The shell arguments ($1 $2 ...) are reset so that each option is preceded by a dash (-); each option argument is also in its own shell argument.

Example

The following code fragment shows you how to process the arguments for a command that can take the **a** and **b** options, and the **o** option, which require an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
     echo $USAGE
     exit 2
fi
for i in $*
do
     case $i in
     -a | -b)    FLAG=$i; shift;;
     -o)         OARG=$2;   shift; shift;;
     - -)        shift;  break;;
     esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg - - file file
```

See Also

getopt(S), sh(C)

Diagnostics

Getopt prints an error message on the standard output when it encounters an option letter not included in *optstring*.

Name

gets - Gets a string from the standard input.

Syntax

gets [*string*]

Description

Gets can be used with csh(C) to read a string from the standard input. If *string* is given, it is used as a default value if an error occurs. The resulting string (either *string* or as read from the standard input) is written to the standard output. If no *string* is given and an error occurs, gets exits with exit status 1.

See Also

csh(C), line(C)

Name

glossary - Defines common UNIX system terms and symbols.

Syntax

[**help**] **glossary** [*term*]

Description

The operating system Help Facility command, **glossary**, provides definitions of common technical terms and symbols.

Without an argument, **glossary** displays a menu screen listing the terms and symbols that are currently included in **glossary**. A user may choose one of the terms or may exit to the shell by typing **q** (for "quit"). When a term is selected, its definition is retrieved and displayed. By selecting the appropriate menu choice, the list of terms and symbols can be redisplayed. Press Retn after entering your choice.

A term's definition may also be requested directly from shell level (as shown in the syntax), causing a definition to be retrieved and the list of terms and symbols not to be displayed. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following table lists the symbols and their escape sequence.

SYMBOL	ESCAPE SEQUENCE
""	\"\"
``	\`\`
[]	\\[\\]
'	\'\'
#	\#
&	\&
*	*
\	\\\\
|	|

From any screen in the Help Facility, a user may execute a command via the shell (sh(C)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level

prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable SCROLL must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your .profile file (see **profile**(M)):

```
export SCROLL ; SCROLL=no
```

If you later decide that scrolling is desired, SCROLL must be set to **yes**.

For information on each of the Help Facility commands, see **help**(C).

See Also

help(C), helpadm(M), locate(C), sh(C), starter(C), usage(C), term(M)

Warnings

If the shell variable TERM (see **sh**(C)) is not set in the user's .profile file, then TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to **term**(M).

Name

graph - Draws a graph.

Syntax

graph [*options*]

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.

If the coordinates of a point are followed by a non-numeric 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 new-lines.

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 a label for the 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 (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

-s Save screen, do not 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(C), tplot(C)

Notes

The terminal you use must have graphics capabilities for successful execution of this command. **Graph** stores all points internally and drops those for which there is no room. Segments that run out of bounds are dropped, not windowed. Logarithmic axes may not be reversed.

Name

grep - Searches a file for a pattern.

Syntax

grep [*options*] *limited regular expression* [*file...*]

Description

Grep searches files for a pattern and prints all lines that contain that pattern. **Grep** uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with **ed**(C) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and **** in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes **'...'**. If no files are specified, **grep** assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it wa found. This can be useful in locating block numbers by context (first block is 0).

-c Print only a count of the lines that contain the pattern.

-i Ignore upper/lower-case distinction during comparisons.

-l Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.

-n Precede each line by its line number in the file (first line is 1).

-s Suppress error messages about nonexistent or unreadable files

-v Print all lines except those that contain the pattern.

See Also

ed(C), egrep(C), fgrep(C), sed(C), sh(C)

Diagnostics

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

Notes

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in /usr/include/stdio.h. If there is a line with embedded nulls, **grep** will only match up to the first null; if it matches, it will print the entire line.

Name

haltsys - Closes out the file systems and halts the CPU.

Syntax

/etc/haltsys

Description

You must be the super-user to access this command.

The **haltsys** command immediately terminates the operating system and should only be used if a system problem prevents the running of **shutdown.** Do not run **haltsys** in multiuser mode and when other users are on the system. Since **haltsys** takes effect immediately, user processes should be killed beforehand (see **kill**(C)).

Related Commands

shutdown(C), kill(C), ps(C)

Name

hd - Displays files in hexadecimal format.

Syntax

hd [*-format*] [**-s** *offset*] [**-n** *count*] [*file ...*]

Description

The **hd** command displays the contents of *file* in hexadecimal, octal, decimal, and character formats. Control over the specification of ranges of characters is also available. The default behavior is with the following flags set: **-abx -A**. This says that addresses (file offsets) and bytes are printed in hexadecimal and that characters are also printed. If no file argument is given, the standard input is read.

Options include:

-s *offset* Specify the beginning *offset* in the file where printing is to begin. If no file argument is given, or if a seek fails because the input is a pipe, offset bytes are read from the input and discarded. Otherwise, a seek error will terminate processing of the current file.

The *offset* may be given in decimal, hexadecimal (preceded by '0x'), or octal (preceded by a '0'). It is optionally followed by one of the following multipliers: w, l, b, or k; for words (2 bytes), long words (4 bytes), blocks (512 bytes), or K bytes (1024 bytes). Note that this is one case where "b" does not stand for bytes. Since specifying a hexadecimal offset in blocks would result in an ambiguous trailing 'b', any offset and multiplier may be separated by an asterisk (*).

-n *count* Specify the number of bytes to process. The *count* is in the same format as *offset*, above.

Format Flags

Format flags may specify addresses, characters, bytes, words (2 bytes), or longs (4 bytes) to be printed in hexadecimal, decimal, or octal. Two special formats may also be indicated: text or ASCII. Format and base specifiers may be freely combined and repeated as desired in order to specify different bases (hexadecimal, decimal or octal) for different output formats (addresses, characters, etc.). All format flags appearing in a single argument are applied as appropriate to all other flags in that argument.

acbwlA Output format specifiers for addresses, characters, bytes, words, longs and ASCII, respectively. Only one base specifier will be used for addresses; the address will appear on the first line of output that begins each new offset in the input.

The character format prints printable characters unchanged, special C escapes as defined in the language, and remaining values in the specified base.

The ASCII format prints all printable characters unchanged, and all others as a period (.). This format appears to the right of the first of other specified output formats. A base specifier has no meaning with the ASCII format. If no other output format (other than addresses) is given, **bx** is assumed. If no base specifier is given, all of **xdo** are used.

xdo Output base specifiers for hexadecimal, decimal and octal. If no format specifier is given, all of **acbwl** are used.

t Print a text file, each line preceded by the address in the file. Normally, lines should be terminated by a \n character; but long lines will be broken up. Control characters in the range 0x00 to 0x1f are printed as `^@' to `^_'. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special characters (^, ~, \) are preceded by a backslash (\) to escape their spe-

cial meaning. As special cases, two values are represented numerically as `\177' and `\377'. This flag will override all output format specifiers except addresses.

Name

hdr - Displays selected parts of object files.

Syntax

hdr [**-dhIKmprsSt**] *file ...*

Description

Hdr displays object file headers, symbol tables, and text or data relocation records in human-readable formats. It also prints out seek positions for the various segments in the object file. Only a.out, x.out, and x.out segmented formats and archives are understood. COFF format files are **not** handled; see **dump**(CP).

The symbol table format consists of six fields. In a.out formats, the third field is missing.

1. The first field is the symbol's index or position in the symbol table, printed in decimal. The index of the first entry is zero.

2. This field is the type, printed in hexadecimal.

3. The third field is the s_seg field, printed in hexadecimal.

4. The fourth field is the symbol's value in hexadecimal.

5. This field is a single character which represents the symbol's type as in **nm**(CP), except C common is not recognized as a special case of undefined.

6. The sixth field is the symbol name.

If long form relocation is present, the format has six fields.

1. The first is the descriptor, printed in hexadecimal.

2. The second is the symbol ID, or index, in decimal. This field is used for external relocations as an index into the symbol table. It should reference an undefined symbol table entry.

3. This field is the position, or offset, within the current segment where relocation is to take place (printed in hexadecimal).

4. The fourth field is the name of the segment referenced in the relocation: text, data, bss or EXT for external.

5. The fifth field is the size of relocation: byte, word (2 bytes), or long.

6. This field, if present, indicates the relocation is relative.

If short form relocation is present, the format has three fields.

1. The first field is the relocation command in hexadecimal.

2. This field has the referenced segment name: text or data.

3. This field indicates the size of relocation: word or long.

Options and their meanings are:

-h Causes the object file header and extended header to be printed out. Each field in the header or extended header is labeled. This is the default option.

-I Uses Intel kernel dataseg (150) and textseg (158) instead of the default 27 and 3F, respectively. (Numbers are in hex.)

-K Uses kernel dataseg (18) and textseg (20) instead of the default 27 and 3F, respectively. (Numbers are in hex.)

-d Causes the data relocation records to be printed out.

-m Prints segment table memory images only.

-p Causes seek positions to be printed out as defined by macros in the include file, a.out.h.

-r Causes both text and data relocation to be printed.

-s Prints the symbol table.

-S Prints the file segment table with a header. (Only applicable to x.out segmented executable files.)

-t Causes the text relocation records to be printed out.

See Also

a.out(F), nm(CP), dump(CP) in the *Reference (CP, S, F)*

Name

head - Prints the first few lines of a stream.

Syntax

head [*-count*] [*file ...*]

Description

This filter prints the first *count* lines of each of the specified *files*. If no files are specified, **head** reads from the standard input. If no *count* is specified, then 10 lines are printed.

See Also

tail(C)

Notes

This utility was developed at the University of California at Berkeley and is used with permission.

Name

help - Operating system Help Facility.

Syntax

help
[**help**] **starter**
[**help**] **usage** [**-d**] [**-e**] [**-o**] [*command_name*]
[**help**] **locate** [*keyword1* [*keyword2*]...]
[**help**] **glossary** [*term*]
help *arg...*

Description

The system Help Facility provides on-line assistance for operating system users, whether they desire general information or specific assistance for use of the Source Code Control System (SCCS) commands.

Without arguments, **help** prints a menu of available on-line assistance commands with a short description of their functions. The commands and their descriptions are:

Command	Description
starter	Information about the operating system for the beginning user
locate	Locate operating system commands using function-related keywords
usage	Operating system command usage information
glossary	Definitions of operating system technical terms

The user may choose one of the above commands by entering its corresponding letter (given in the menu), or may exit to the shell by typing **q** (for "quit").

With arguments, **help** directly invokes the named on-line assistance command, bypassing the initial **help** menu. The commands **starter, locate, usage,** and **glossary,** optionally preceded by the word **help,** may also be specified at shell level. When executing **glossary** from shell level some of the symbols listed in the glossary must be escaped (preceded by one or more backslash (\) characters) to be understood by the Help Facility. For a list of symbols and how many backslashes to use for each, refer to the **glossary**(C) manual page.

From any screen in the Help Facility, a user may execute a command via the shell (**sh**(C)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable SCROLL must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your .profile file (see **profile**(M)):

export SCROLL ; SCROLL=no

If you later decide that scrolling is desired, SCROLL must be set to **yes.**

Information on each of the Help Facility commands (**starter, locate, usage, glossary,** and **help**) is located on their respective manual pages.

If the first argument to **help** is different from **starter, usage, locate,** or **glossary, help** assumes information is being requested about the SCCS Facility. The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type1 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., ge3 for message 3 from the **get**(C) command).

type2 Does not contain numerics (as a command, such as **get**).

type3 Is all numeric (e.g., 212).

See Also

glossary(C), locate(C), sh(C), starter(C), usage(C), term(M) profile(M), and admin(S), cdc(S), comb(S), delta(S), get(S), prs(S), rmdel(S), sact(S), sccsdiff(S), unget(S), val(S), vc(S), what(S), sccsfile(F) in the *Reference (CP, S, F)*

Warnings

If the shell variable TERM (see **sh**(C)) is not set in the user's .profile file, then TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to **term**(M).

Name

hplp - Filters files for printing on HP Laserjet.

hplpR - Filters and reverses pages for printing on HP Laserjet.

Syntax

hplp [*file ...*]
hplpR [-w] [*file ...*]

Description

These commands filter files for printing on the Hewlett-Packard Laserjet printer. If no *files* are given, both commands will read from the standard input. Both commands write the output to the standard output (screen), and are normally run as part of a pipeline:

hplp *file* | **lpr***N*

Hplp simply prepends the command sequences that enable the printing of a full 66 lines by 80 columns on standard 8 1/2 x 11 paper (without putting lines in the "unprintable" regions of the paper).

HplpR will reverse the pages in a file, so that when they are actually printed, they will be correctly collated in the output tray. A maximum of 256 pages can be reversed. It is assumed that all pages are 66 lines, so documents formatted for other page lengths may not be handled correctly. If a formfeed (octal 014) is found, it terminates a page, allowing correct reversal of short pages.

-w Prints wide documents (**hplpR**). The -w option sends the command sequences that request "landscape mode" printing (rotated 90 degrees), and that use 17 pitch characters. This allows printing of pages with a full 66 lines by 170 columns.

Files

/usr/bin/hplp

(BLANK)

Name

id - Prints user and group IDs and names.

Syntax

id

Description

Id outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

See Also

getuid(S), logname(C)

Name

ipcrm - Removes a message queue, semaphore set, or shared memory id.

Syntax

ipcrm [*options*]

Description

Ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

-m *shmid* Removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-M *shmkey* Removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-q *msqid* Removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

-Q *msgkey* Removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

-s *semid* Removes the semaphore identifier *semid* frc the system and destroys the set of semaphores and data structure associated with it.

-S *semkey* Removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in **msgctl**(S), **shmctl**(S), **semctl**(S). The identifiers and keys may be found by using **ipcs**(C).

See Also

ipcs(C) and msgctl(S), msgget(S), msgop(S), semctl(S), semget(S), semop(S), shmctl(S), shmget(S), shmop(S) in the *Reference (CP, S, F)*

Name

ipcs - Reports inter-process communication facilities status.

Syntax

ipcs [*options*]

Description

Ipcs prints certain information about active inter-process communication facilities. Without options, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following options.

Options

-m Print information about active shared memory segments.

-q Print information about active message queues.

-s Print information about active semaphores.

If any of the options **-m**, **-q**, or **-s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed subject to these options:

-a Use all print options. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

-b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See the following for meaning of columns in a listing.

-c Print creator's login name and group name. See the following description.

-C *corefile*
Use the file *corefile* in place of **/dev/kmem.**

-N *namelist*
The argument will be taken as the name of an alternate *namelist* (/unix is the default).

-o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

-p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments). See the following description.

-t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop* on semaphores.) See the following description.

The column headings and the meaning of the columns in an **ipcs** listing follow; the letters in parentheses indicate the options that cause the corresponding heading to appear; "all" means that the heading always appears. Note that these options only determine what information is provided for each facility; they do not determine which facilities will be listed.

T (all)
Type of the facility:
m Shared memory segment;
q Message queue;
s Semaphore.

ID (all)
The identifier for the facility entry.

KEY (all)

The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.)

MODE (all)

The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

C if the associated shared memory segment is to be cleared when the first attach is executed;

D if the associated shared memory segment has been removed. It will disappear when the last process attached t the segment detaches it;

R if a process is waiting on a *msgrcv*;

S if a process is waiting on a *msgsnd*;

\- if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

a if alter permission is granted;

r if read permission is granted;

w if write permission is granted;

- if the indicated permission is not granted.

OWNER (all)
The login name of the owner of the facility entry.

GROUP (all)
The group name of the group of the owner of the facility entry.

CREATOR (a,c)
The login name of the creator of the facility entry.

CGROUP (a,c)
The group name of the group of the creator of the facility entry.

CBYTES (a,o)
The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o)
The number of messages currently outstanding on the associated message queue.

QBYTES (a,b)
The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p)
The process ID of the last process to send a message to the associated queue.

LRPID (a,p)
The process ID of the last process to receive a message from the associated queue.

STIME (a,t)
The time the last message was sent to the associated queue.

RTIME (a,t)
The time the last message was received fror the associated queue.

CTIME (a,t)
The time when the associated entry was created or changed.

NATTCH (a,o)
The number of processes attached to the associated shared memory segment.

SEGSZ (a,b)
The size of the associated shared memory segment.

CPID (a,p)
The process ID of the creator of the sharec memory entry.

LPID (a,p)
The process ID of the last process to attach or detach the shared memory segmen

ATIME (a,t)
The time the last attach was completed to the associated shared memory segment.

DTIME (a,t)
The time the last detach was completed on the associated shared memory segment.

NSEMS (a,b)
The number of semaphores in the set associated with the semaphore entry.

OTIME (a,t)
The time the last semaphore operation was completed on the set associated with the semaphore entry.

Files

/unix	System namelist
/dev/kmem	Memory
/etc/passwd	User names
/etc/group	Group names

See Also

msgop(S), semop(S), shmop(S) in the *Reference Manual (CP, S, F)*

Notes

Things can change while **ipcs** is running; the picture it gives is only a close approximation to reality.

Name

join - Joins two relations.

Syntax

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 (see **sort**(C)).

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*.

The default input field separators are blank, tab, or newline. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the options below use the argument *n*. This argument should be a 1 or a 2 referring to either *file1* or *file2*, respectively. The following options are recognized:

-a*n* 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*.

-j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with 1.

-o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.

-t*c* Use character *c* as a separator. Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

Example

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group

See Also

awk(C), comm(C), sort(C), uniq(C)

Notes

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**(C), **sort**(C), **comm**(C), **uniq**(C), and **awk**(C) are wildly incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

(BLANK)

Name

kill - Terminates a process.

Syntax

kill [*-signo*] *PID...*

Description

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using **ps**(C).

The details of the **kill** are described in **kill**(S). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user. If a signal number preceded by - is given as the first argument, that signal is sent instead of terminate (see **signal**(S)). In particular **kill -9** *PID* is a sure kill.

See Also

ps(C), sh(C), and kill(S), signal(S) in the *Reference (CP, S, F)*

Name

killall - Kills all active processes.

Syntax

/etc/killall [*signal*]

Description

Killall terminates all active processes not directly related to the shutdown procedure. **Killall** is used by /etc/shutdown, and can only be run by the super-user.

Killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* (see **kill**(C)). The default signal is 9.

Files

/etc/shutdown

See Also

kill(C), ps(C), shutdown(C)

Name

labelit - Provides labels for file systems.

Syntax

/etc/labelit *special* [*fsname volume* [**-n**]]

Description

Use **labelit** to provide labels for unmounted disk file systems or file systems being copied to tape.

-n Provides for initial labeling only (this destroys previous contents).

With the optional arguments omitted, **labelit** prints current label values.

The *special* name should be the physical disk section (e.g., /dev/hd0b), or the cartridge tape (e.g., /dev/rct). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (e.g., /, /usr2, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette or tape.

For file systems on disk, *fsname* and *volume* are recorded in the superblock.

See Also

sh(C)

Name

last - Indicates last logins of users and terminals.

Syntax

last [**-f** *file*] [**-t tty***nn...*] [*name...*]

Description

The **last** command looks in the wtmp file (where every login and logout is recorded) for information about a user, a terminal, or any group of users and terminals. Other arguments specify names of users or terminals.

The **last** command displays the sessions of the specified users and terminals, most recent first, indicating the times the session began, the duration, and terminals used. The **last** command indicates if the session was cut short by a reboot. There is a pseudo-user "reboot" that is logged in each time the system reboots. So the command:

last reboot

gives an indication of mean time between reboot.

For multiple arguments, information applying to any of the arguments is printed. For example,

last root croot

lists all of root's sessions as well as all of croot's sessions.

last -t tty02 console

lists all logins on tty02 and console.

If the **last** command is issued with no arguments, a record of all logins and logouts are displayed in reverse order.

Options

-f *file* Specifies an alternate wtmp file.

-t tty*nn* Lists the logins on the named terminal (separate terminal names with a space).

Related Commands

utmp(M)

See Also

/usr/adm/wtmp Login data base

Name

leave - Reminds you when you have to leave.

Syntax

leave [*hhmm*]

Description

Leave waits until the specified time, then reminds you that you have to leave. You are reminded five minutes and one minute before the actual time, at the time, and every minute thereafter. When you log off, **leave** exits just before it would have printed the next message.

The time of day is in the form *hhmm* where *hh* is a time in hours (on a 12 or 24 hour clock). All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, **leave** prompts with "When do you have to leave?" A reply of newline causes **leave** to exit, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a .login or .profile.

Leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use "kill -9" giving its process id.

See Also

calendar(C)

Name

line - Reads one line of input.

Syntax

line

Description

Line copies one line (up to a newline) from the standard input and writes it on the standard output. It returns an exit code of 1 on end-of-file and always prints at least a newline. It is often used within shell files to read from the user's terminal.

See Also

gets(C), sh(C), and read(S) in the *Reference (CP, S, F)*

Name

ln - Makes a link to a file.

Syntax

ln [*options*] *file1 file2*
ln [*options*] *file1 ... filen directory*

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 to the file are effective independent of the name by which the file is known.

Ln creates a link to the existing file, *file1*. The *file2* argument is a new name referring to the same file contents as *file1*. If the last argument is a directory, links to *file1 ... filen* will be made in *directory*.

Ln has the following options:

-f Makes the link even if *file2* already exists (by first unlinking *file2*).

-s Makes a symbolic link to a file. A symbolic link differs from a regular link in that it is a separate inode on disk that points to another file. The target of a symbolic link may be on a different file system, or even on a different machine if the network is in use.

Symbolic links can nest three deep. That is, if W is a file, X may be symbolically linked to W, Y to X, and Z to Y. However, if ZZ is symbolically linked to Z, attempting to access ZZ will fail.

See Also

cp(C), ls(C), mv(C), rm(C), and symlink(S) in the *Reference (CP, S, F)*

Notes

You cannot make a hard link to a directory or across file systems.

Name

login - Gives you access to the system.

Syntax

login [*name* [*env-var...*]]

Description

The **login** command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user logs out by typing a Ctrl-d.

If **login** is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure /etc/profile is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command inter-

preter (usually **sh**(C)) are initialized, and the file .profile in the working directory is executed, if it exists.

These specifications are found in the /etc/passwd file entry for the user. The name of the command interpreter is - followed by the last component of the interpreter's path name (e.g., -sh). If this field in the password file is empty, then the default command interpreter, /bin/sh is used. If this field is "*", then the named directory becomes the root directory, the starting point for path searches for path names beginning with a /. At that point, **login** is re-executed at the new level which must have its own root structure, including /etc/login and /etc/passwd.

The basic environment is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment may be expanded or modified by supplying additional arguments to **login**, either at execution time or when **login** requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as:

L*n*=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables PATH and SHELL cannot be changed. This prevents users, logging into restricted shell environments, from spawning secondary shells that are not restricted. Both **login** and **getty**(M) understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

Files

/etc/utmp	Accounting
/etc/wtmp	Accounting
/usr/mail/*your-name*	Mailbox for user *your-name*
/etc/motd	Message-of-the-day
/etc/passwd	Password file
/etc/profile	System profile
profile	User's login profile

See Also

mail(C), newgrp(C), sh(C), environ(M), getty(M), su(M), passwd(M), profile(M)

Diagnostics

login incorrect
The user name or the password cannot be matched.

No shell, cannot open password file, or *no directory*
There is an error in the password file, /etc/passwd.

No utmp entry. You must exec "login" from the lowest level shell
You attempted to execute **login** as a command without using the shell's internal **exec** command or from a subshell.

Name

logname - Gets login name.

Syntax

logname

Description

Logname returns the user's login name as set when the user logs into the system.

See Also

env(C), environ(M), login(C), and logname(S) in the *Reference (CP, S, F)*

Name

look - Finds lines in a sorted list.

Syntax

look [**-df**] [**-t***c*] *string* [*file*]

Description

Look consults a sorted file and prints all lines that begin with *string*. It uses binary search.

The **-d** and **-f** options affect comparisons as in **sort**(C):

-d Dictionary order: only letters, digits, tabs, and spaces are compared.

-f Fold: uppercase letters compare equally to lowercase letters.

-t*c* Specify an alternate tab character (word separator), *c*.

If no file is specified, /usr/dict/words is assumed with the collating sequence, **-df**.

Files

/usr/dict/words

See Also

grep(C), sort(C)

Name

lp, cancel - Sends/cancels requests to an LP line printer.

Syntax

lp [**-c**] [**-d***dest*] [**-m**] [**-n***number*] [**-o***option*]
[**-s**] [**-t***title*] [**-w**] *file ...*
cancel [*ids*] [*printer ...*]

Description

Lp arranges for the named *files* and associated information (collectively called a request) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name - (the standard input) may also be supplied on the command line along with named files. The order in which the files appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see **cancel** below) or find the status (see **lpstat**(C)) of the request.

Options

-c Make copies of the files to be printed immediately when **lp** is invoked. For RFS, you must specify **-c**, which copies the files to /usr/spool/lp/request/*printername*. Normally, the *files* will not be copied, but will be linked whenever possible. If the **-c** option is not given, then be careful not to remove any of the files before the request has been printed in its entirety. It should also be noted that in the absence of the **-c** option, any changes made to the named files after the request is made but before it is printed will be reflected in the printed output.

NOTE

You must specify **-c** when printing across an RFS network.

-d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer in that class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted. By default, *dest* is taken from the environment variable LPDEST (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems.

-m Send mail (see **mail**(C)) after printing the files. By default, no mail is sent upon normal completion of the print request.

-n*number* Print *number* copies (default of 1) of the output.

-o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **-o** keyletter more than once.

-s Suppress messages from **lp** such as "request id is ...."

-t*title* Print *title* on the banner page of the output.

-w Write a message on the user's terminal after printing the files. If the user is not logged in, sends mail instead.

Cancel cancels line printer requests that were made by the **lp** command. The command line arguments may be either request *ids* (as returned by **lp**) or printer names (see **lpstat** for a list of names). Specifying **a request** *id* cancels the associated request even if it is currently printing. Specifying a printer cancels the request which is cur-

rently printing on that printer. In either case, cancelling a request that is curerntly printing frees the printer to print its next available request.

If you attempt to print with **lp** a file that is not readable by others or is in a directory not accessible by others, then you must use one of the following syntaxes to print such a file with **lp:**

cat *file* | **lp**
lp < *file*

If your system uses **lpr** instead of **lp,** these restrictions do not apply.

Files

/usr/spool/lp/*

See Also

lpadmin(M), lpenable(C), lpinit(M), lpstat(C), mail(C), accept(C), lpsched(M), *Operations Guide*

Name

lpenable, lpdisable - Enables/disables LP printers.

Syntax

lpenable *printer ...*
lpdisable [**-c**] [**-r**[*reason*]] *printer ...*

Description

Lpenable activates the named printers to print requests taken by **lp**(C). Use **lpstat**(C) to find the status of the printers.

Lpdisable deactivates the named printers, disabling them from printing requests taken by **lp**(C). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use **lpstat**(C) to find the status of printers.

Options useful with **lpdisable** are:

-c Cancel any requests that are currently printing on any of the designated *printers*.

-r[*reason*] Associates a *reason* with the deactivation of the *printers*. This reason applies to all printers mentioned up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a *reason*, then a default reason will be used. *Reason* is reported by **lpstat**(C).

Files

/usr/spool/lp/*

See Also

lp(C), lpstat(C)

Name

lpr - Routes named files to printer spooler.

Syntax

lpr [**-b**[*x*]] [*-k*] [**-m**[*login*]] [**-n**] [**-@**[*netname*]] [**-p***k*] [**-r**] [**-s***conf*] [**-S***modes*] *file name(s)*

Description

The **lpr** command routes the named files to the printer you specify. (See "Notes" for restrictions.) There are several options that you can add to the **lpr** command so it will do more than route the files to a printer.

Most of these options have an associated environment variable and a corresponding field in a configuration file line (see **printers**(M)). Command-line arguments have precedence over environment variables, and environment variables override configuration file fields.

NOTE

Beginning with Altos System V, Release 5.3d, the **lpr** program is a filter for (calls) the **lp** program. It no longer processes the -N and -S options. See the description of **lp** in the *Operations Guide* for information on how to set up a printer interface program.

Options

-b*x* Adds a banner page at the beginning of a print job. The *x* argument (4 characters maximum) supplies the text of the header. Use of this option overrides the BANNER environment variable, which tells **lpd** and **lpr** that BANNER is the name used in the header page attached to your job.

If you use the **-b** option without defining *x*, and the BANNER environment variable is undefined, the spooler will use your login name.

If you specify **-b**, the banner page is printed, then the print job, and a formfeed is supplied after the print job is finished. If no banner is desired, printing begins at the current position of the printhead (no initial formfeed), and a page is ejected after printing is completed.

-m[*login*] Sends you mail when a printer job has completed. If you specify a login name, the specified user is notified; otherwise, the **-m** option alone sends a message to the requestor.

-N Suppresses the formfeed after each page.

-@[*netname*] Specifies a WorkNet computer name other than your local machine.

When you use this option, the files you want to print are copied to spool directories on the named machine. A print daemon (see the **lpd** command) is remotely invoked to do the actual printing. If no machine name is specified, printing is assumed to be local.

-p*n* Where *n* is a number that represents the printer device. If you don't use this option, the print spooler checks the environmental variable PRINTER for the printer device number. If PRINTER is not defined, the spooler checks the configuration file.

The default printer spool directory is /usr/spool/lpd and the default printer device is /dev/lp. If printer 1 is selected, the spool directory is /usr/spool/lpd1, and device /dev/lp1.

As an alternative, you can supply the printer number as part of the **lpr** name, (for example, **lpr1** [*-option1*] [*option2*] ...). The **-p** option overrides a printer digit supplied from the file name.

-r Removes a file from the home directory after printing.

-s*conf* Selects a printer configuration line from the printer configuration file (see the miscellaneous file printers in the next section). *Conf* is the name of the configuration line you select from the configuration file. You can also use the environmental variable PCONF to select the configuration line you want.

-S*modes* Supplies baud and other tty modes for serial printers. The *modes* argument consists of a set of tty modes, enclosed in quotes. For example, **-S1200** selects 1200 as the baud rate for this print request. This option overrides a mode selection from the configuration file. You can also use the environment variable PMODES to supply this information.

If you specify a configuration line name, and no printer device, the print spooler uses the first line in the printer configuration file that matches that name. The spooler then takes the printer device digit from the corresponding field in the same line.

If you specify a configuration line name and a printer device, the print spooler uses the first line in the printer configuration file that matches both printer device and name fields.

If you specify a printer device only, or if the spooler is invoked as "lpr," the first line that matches the printer device is used. The printer device defaults to /dev/lp if no configuration file lines are selected, or if no configuration file exists.

Examples

This command prints the file lemons on the local printer, and sends you a message when the print job is finished.

lpr -m lemons Retn

This command queues the file lemons to be printed from spool directory /usr/spool/lpd2 on printer /dev/lp2 at a speed of 1200 baud.

lpr -S1200 -p2 lemons Retn

This command prints the file lemons on the default printer of the computer machine1 in your network, and sends user chris a message when the print job is finished.

lpr -mchris -@machine1 lemons Retn

Files

/usr/spool/lpd*	Spooling directories
/dev/lp*	Line printer devices
/usr/lib/lpd	Line printer daemon
/etc/printers	Printer configuration file

See Also

mail(C), lpd(M), lp(C), printers(M)

Notes

For Unix System V compatibility on the Altos 386 Series 500 and software release 5.3d and later, **lpr** executes **lp** and its associated commands.

Name

lpstat - Prints LP status information.

Syntax

lpstat [*options*] [*requestid...*]

Description

Lpstat prints information about the current status of the LP line printer system.

If no options are given, then **lpstat** prints the status of all requests made to **lp**(C) by the user. Any argument that is not an option is assumed to be a *requestid*, which is a unique id that **lp**(C) associates with each request. **Lpstat** prints the status of such requests.

Options

Options may appear in any order and may be repeated and mixed with other arguments. Some of the keyletters below may be followed by an optional list that can be in one of two forms:

- A list of items separated from one another by a comma
- A list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

For example:

-u"user1, user2, user3"

The omission of a list following such keyletters causes all information relevant to the keyletter to be printed. For example:

lpstat -o

prints the status of all output requests.

All options listed below apply to a Worknet network. In addition, local/remote refers to local/remote machines in an RFS network.

-a[*list*] Print acceptance status (with respect to **lp**) of destinations for requests. *List* is a list of mixed printer names and class names. RFS local/remote.

-c[*list*] Print class names and their members. *List* is a list of class names. RFS local.

-d Print the system default destination for **lp**. The destination can be local/remote.

-o[*list*] Print the status of output requests. *List* is a list of mixed printer names, class names, and request ids. RFS local/remote.

-p[*list*] Print the status of printers. *List* is a list of printer names. RFS local/remote.

-r Print the status of the LP request scheduler. RFS local.

-s Print a status summary, including the system default destination, a list of class names and their members, and a list of printers and their associated devices. RFS local.

-t Print all status information. RFS local.

-u[*list*] Print status of output requests for users. *List* is a list of login names. RFS local.

-v[*list*] Print the names of printers and the path names of the devices associated with them. *List* is a list of printer names. RFS local/remote.

Files

/usr/spool/lp/*

See Also

lpenable(C), lp(C)

Name

ls - Lists contents of a directory.

Syntax

ls [*options*] [*name ...*]
l
lc
lf
ll
lr
lx

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. **Lc, lf, ll, lr,** and **lx** abbreviate **ls -C, ls -F, ls -l, ls -R** and **ls -x,** respectively.

The **ls** command has the following options:

-A List all entries including those that begin with a dot except "." and "..".

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-b Force printing of non-printing characters to be in the octal *\ddd* notation.

-C Multi-column output with entries sorted down the columns.

-c Use time of creation of the inode for sorting (**-t**).

-d If an argument is a directory, list only its name (not its contents); often used with **-l** to get the status of a directory.

-F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable. This is the same as typing **lf**.

-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 The same as **-l**, except that the owner is not printed.

-i For each file, print the inode number in the first column of the report.

-L Mark directories or files with a trailing ">" if they are symbolic links and the **-l** option is not used. If **-l** is used, list the name of the file to which it is symbolically linked.

-l List in long format, giving mode, number of links, owner, group, 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 rather than a size.

-m Stream output format; files are listed across the page, separated by commas.

-n The same as **-l**, except that the owner's UID and group's GID numbers are printed, rather than the associated character strings.

-o The same as **-l**, except that the group is not printed.

-p Put a slash (/) after each filename if that file is a directory.

-q Force printing of non-printing characters in file names as the character (?).

-R Recursively list subdirectories encountered.

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-s Give size in blocks, including indirect blocks, for each entry.

-t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See -u and -c.)

-u Use time of last access instead of last modification for sorting (with the -t option) or printing (with the -l option).

-x Multi-column output with entries sorted across rather than down the page.

-1 List only one (1) entry per line, even if output is to a terminal. This is the default when output is not to a terminal.

The mode printed under the -l option consists of ten characters. The first character may be one of the following:

b the entry is a block special file
c the entry is a character special file
d the entry is a directory
l the entry is a symbolic link
m the entry is a shared memory special file
p the entry is a fifo (a.k.a. "named pipe") special file
s the entry is a semaphore file
- the entry is an ordinary file

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

ls -l (the long list) prints its output as follows:

```
-rwxrwxrwx  1 smith  dev     10876  May 16 9:42 part2
```

This horizontal configuration provides a good deal of information. Reading from right to left, you see that the current directory holds one file, named "part2." Next, the

last time that file's contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group "dev" (perhaps indicating "development"), and his or her login name is "smith." The number, in this case "1," indicates the number of links to file "part2." Finally, the row of dashes and letters tell you that user, group, and others have permissions to read, write, execute "part2."

The execute (x) symbol here occupies the third position of the three-character sequence. A - in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

r The file is readable

w The file is writable

x The file is executable

- The indicated permission is *not* granted

s The set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on

S The set-user-ID bit or set-group-ID bit is on and the corresponding execution bit is off

t The 1000 (octal) bit, or sticky bit, is on (see **chmod**(C)), and execution is on

T The 1000 bit is turned on, and execution by others is off

For user and group permissions, the third position is sometimes occupied by a character other than x or -. s also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the file owner during execution is, for example, used by the **passwd** command to allow you to update your password file entry, normally writeable only by the super-user.

For others permissions, the third position may be occupied by t or T. These refer to the state of the sticky bit and execution permissions.

Examples

The first set of examples refers to permissions:

-rwxr--r--

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

-rwsr-xr-x

The second example describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

ls -a

This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

ls -aisn

This command will provide you with quite a bit of information including **a**ll files, including non-printing ones (**a**), the **i**-number--the memory address of the i-node associated with the file--printed in the left-hand column (**i**); the size (in blocks) of the files, printed in the column to the right of the inumbers (**s**); finally, the report is displayed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

Files

/etc/passwd	User IDs for **ls -l** and **ls -o**
/etc/group	Group IDs for **ls -l** and **ls -g**
/usr/lib/terminfo/?/*	Terminal information database

See Also

chmod(C), find(C)

Notes

Unprintable characters in file names may confuse the columnar output options.

Name

mail - Sends or receives mail among users.

Syntax

mail [*user ...*]
mail [**-f** *file*] [**-pr**]

Description

The **mail** command (/bin/mail) is used both to send mail to other users, and to read mail that has been sent to you. When you log in, the system tells you if you have mail.

There are several options you can use with the **mail** command to print, delete, and store mail.

Options

When you see the "You have mail" message, read your mail by entering:

mail Retn

You can use any of the following arguments with the **mail** command:

-f [*file*]	Causes the named *file* to be displayed in mail format.
-p	Causes the mail to be printed on your printer.
-r	Causes the oldest message to appear first. Without this option, the most current message appears first.

You can enter the following at the "?" prompt:

Retn	Goes on to the next message
d	Deletes the current message and goes on to the next one

Ctrl-d (EOF)	Puts unexamined mail back in the mailbox and stops
m[*user ...*]	Sends a copy of the current message to the specified users
p	Displays a message again
q	Puts mail back in the mailbox and quits
s [*file ...*]	Saves the message in the named file(s)
w [*file ...*]	Saves the message, without a header, in the named file(s)
x	Exits, without changing the mailbox file
-	Goes back to previous message
+	Goes to next message. After last message, returns you to system prompt.
!*command*	Escapes to the shell and executes the specified command
?	Displays a summary of what you can enter at the "?" prompt

Examples

This command sends the message "This is a message sent to chris, wendy, and ric" to chris, wendy, and ric. Press Retn, then Ctrl-d, to send a message.

mail chris wendy ric Retn
This is a message sent to chris, wendy, and ric. Retn
Ctrl-d

This command prints your mail on your system's printer.

mail -p Retn

Files

/usr/spool/mail/*	Mailboxes
/etc/passwd	Identifies senders and recipients
mbox	Saved mail
/tmp/ma*	Tempfile
dead.letter	Unmailable text

Name

mail - Sends, reads, or disposes of mail.

Syntax

mail [[**-u** *user*] [**-b** *bcclist*] [**-c** *cclist*] [**-r** *rrlist*]
[**-f** *mailbox*]] [**-d**] [**-e**] [**-R**] [**-i**] [*user ...*]
mail [**-s** *subject*] [**-i**] [*user ...*]

Description

Mail (/usr/bin/mail) is a mail processing system that supports composing of messages, and sending and receiving of mail between multipe users. When sending mail, a *user* is the name of a user or of an alias assigned to a machine or to a group of users.

Options include:

-u *user* — Tells **mail** to read the system mailbox belonging to the specified *user*.

-b *bcclist*
-c *cclist*
-r *rrlist* — When sending mail, initialize the bcc (before carbon copy), cc (carbon copy), or rr (return receipt) fields in the message header to the following list of users. These fields may be subsequently added to or edited through the use of **~bcc**, **~cc**, **~rrt**, and **~header** commands in compose mode. Only the last occurrence of each flag is used.

-f *mailbox* — Tells **mail** to read the specified *mailbox* instead of the default user's system mailbox.

-d — Prints debugging information.

-e — Allows escapes from compose mode when input comes from a file.

-R Makes the mail session "read-only" by preventing alteration of the mailbox being read. Useful when accessing system-wide mailboxes.

-i Tells **mail** to ignore interrupts sent from the terminal. This is useful when reading or sending mail over telephone lines where "noise" may produce unwanted interrupts.

-s *subject* Specifies *subject* as the text of the Subject: field for the message being sent.

Sending Mail

To send a message to one or more other people, invoke **mail** with arguments which are the names of peole to send to. You are then expected to type in your message, followed by a Ctrl-d at the beginning of a line.

Reading Mail

To read mail, invoke **mail** with no arguments. This will check your mail out of the system-wide directory so that you can read and dispose of the messages sent to you. A message header is printed out for each message in your mailbox. The current message is initially the last numbered message 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**(C), with the + and - commands moving backwards and forwards, and simple numbers typing the addressed message.

If new mail arrives during the mail session you can read in the new messages with the **restart** command.

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. Usually, though, deleted messages will disappear completely.

Specifying Messages

Commands such as **print** and **delete** often 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 **top** command, 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 sent it. Then you can type the text of the reply, and press Ctrl-d to send it. While you are composing a message, **mail** treats lines beginning with a tilde (~) as special. For instance, typing **~m** (alone on a line) places a copy of the current message into the response, shifting it right by one tabstop. Other escapes set up subject fields, add and delete recipients to the message, and allow you to escape either to an editor to revise the message or to a shell to run some commands. (These options are summarized later.)

Ending a Mail Session

You can end a **mail** session with the **quit(q)** command. Messages that have been examined go to your mbox file unless they have been deleted. 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 **-i** option causes **mail** to ignore interrupts.

Using Aliases and Distribution Lists

It is also possible to create a personal distribution list. For instance, you can send mail to "cohorts" and have it go to a group of people. Such lists can be defined by typing a line like:

alias cohorts bill bob barry bobo betty beth bobbi

in the .mailrc file of your home directory. The current list of such aliases can be displayed by the **alias(a)** command in **mail**. System-wide distribution lists can be created by editing /usr/lib/mail/aliases; see **aliases**(M); these are kept in a slightly different syntax. In mail that 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.

Mail has a number of options that can be set in the .mailrc file to alter its behavior; thus, **set askcc** enables the "askcc" feature. (These options are summarized below.)

Summary

Each **mail** 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 the commands that take message lists as arguments, if no message list is given, then the next message forward that 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 messages at all, **mail** displays "No applicable messages" and aborts the command.

- [*n*]	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.
+ [*n*]	Goes to the next message and prints it out. If given a numeric argument, *n*, goes to the *n*th next message and prints it.
Retn	Goes to the next message and prints it out.
?	Prints a brief summary of commands.
!*command*	Executes the shell command which follows.
=	Prints out the current message number.

^	Prints out the first message.
$	Prints out the last message.
alias (a)	With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, adds the users named in the second and later arguments to the alias named in the first argument.
Alias *users*	Prints system-wide list of aliases for *users*. At least one user must be specified.
cd (c)	Changes the user's working directory to that specified. If no directory is given, it changes to the user's login directory.
delete (d)	Takes a list of messages as an argument and marks them all as deleted. Deleted messages are not retained in the system mailbox after a quit, nor are they available to any command other than the **undelete** command.
dp	Deletes the current message and prints the next message. If there is no next message, **mail** says "no more messages."
echo Path	Expands shell metacharacters.
edit (e)	Takes a list of messages and invokes the text editor on each one in turn. Upon returning from the editor, the message is read back in.
exit (x)	Effects an immediate return to the shell without modifying the user's system mailbox, mbox file, or edit file in **-f**.
file (fi)	Prints the name of the file mail is reading. If it is a mailbox, the name of the owner is returned.
forward (f)	Forwards the current message to the named users. The current message is indented within the forwarded message.

Forward (F) Forwards the current message to the named users. The current message is not indented within the forwarded message.

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. Both **+** and **-** may take a number to view a particular window. If a message-list is given, it prints the specified headers.

hold (ho) Takes a message list and marks each message to be saved in the user's system mailbox instead of in mbox. Use only when the **autombox** switch is set. Does not override the **delete** command.

list Prints a list of **mail** commands.

lpr (l) Prints out each message in a message-list on the lineprinter.

mail (m) Takes as argument login names and distribution group names and sends mail to those people.

mbox (mb) Marks messages in a message list so that they are saved in the user mailbox after leaving mail.

move *mesg-list mesg-num*
Places the messages specified in *mesg-list* after the message specified in *mesg-num*. If *mesg-num* is 0, *mesg-list* moves to the top of the mailbox.

next (n) Goes to the next message in sequence and prints it. With an argument list, **next** types the next matching message (like **+** or Retn).

print (p) Prints out each message in a message-list on the terminal display.

quit (q) Terminates the session, retaining all undeleted, unsaved messages in the system mailbox and removing all other messages. Files marked with a star (*) are saved; files marked with an "M" are saved in the user mailbox. If new mail has arrived during the session, the message "You have new mail" is given. If **quit** is given while editing a mailbox file with the **-f** flag, then the edit file is rewritten. You then return to the shell, unless the rewrite of the edit file fails, in which case you can escape with the **exit** command.

reply (r) Takes a message list and sends mail to each message author. The default message must not be deleted.

Reply (R) Takes a message list and sends mail to each message author and each member of the message just like the **mail** command. The default message must not be deleted.

restart Reads in messages that arrived during the current mail session.

save (s) Takes a message list and a filename and appends each message in turn to the end of the file. The filename, in quotation marks, 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 an option. Arguments are of the form *option=value* or *option.*

shell (sh) Invokes an interactive version of the shell.

size (si) Takes a message list and prints out the size in characters of each message.

source (so) *file*
Reads mail commands from the *file* given as its only argument.

string *string mesg-list*
Searches for *string* in *mesg-list*. If no *mesg-list* is specified, all undeleted messages are searched. Case is ignored in the search.

top (t) Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the *toplines* variable and defaults to six.

undelete (u) Takes a message list and marks each one as *not* being deleted.

unset (uns) Takes a list of option names and discards their remembered values; the inverse of **set**.

visual (v) Takes a message list and invokes **vi** on each message.

whois Looks up a list of target mail recipients and prints the real names or descriptions of each recipient. If the first character of the first argument is alphabetic, the arguments are looked up without change. Otherwise, the arguments are assumed to be a message list. For each message in the list, the "From" person is extracted from the header and added to list of users to be searched.

If a target mail recipient contains a machine and user name, nothing is printed. If it is a private alias, "private alias" is printed. If it is a global alias, the name or description of the recipient is printed (contents of the $n field in the alias file). If all of the above fail, the user is looked up in /etc/passwd; if the user is a local user, "local user" is printed. Finally, if none of the above tests and searches succeed, "unknown" is printed.

write (w) *filename*
Saves the body of the message in the named file.

Here is a summary of the compose escapes, which are used when composing messages to perform special functions. Compose escapes are only recognized at the beginning of lines.

~~*string*	Inserts the string of text in the message prefaced by a single tilde (~). If you have changed the ESCAPE character, then you should double that character instead.
~?	Prints out help for compose escapes.
~.	Same as Ctrl-d on a new line.
~!*cmd*	Executes the indicated shell command, then returns to the message.
~|*cmd*	Pipes the message through the command as a filter. If the command gives no output or terminates abnormally, retains the original text of the message.
~_*mail-command*	Executes a mail command, then returns to compose mode.
~:*mail-command*	Executes a mail command, then returns to compose mode.
~alias	Prints list of private aliases.
~alias *aliasname*	Prints names included in private *aliasname*.
~Alias	Performs aliasing by first examining private aliases and then system-wide aliases using all three global alias files (aliases.hash, faliases, and mailiases). Only the final result is printed (non-local mail recipients will have the complete delivery path printed). The user list is taken from header fields.

~Alias *users* Performs aliasing by first examining private aliases and then system-wide aliases using all three global alias files (aliases.hash, faliases, and mailiases). Only the final result is printed (non-local mail recipients will have the complete delivery path printed). At least one user must be specified.

~b *name...* Adds the given names to the list of blind carbon copy recipients.

~c *name...* Adds the given names to the list of carbon copy recipients.

~cc *name...* Same as ~c.

~d Reads the dead.letter file from your home directory into the message.

~e Invokes the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~h Edits the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field with the current terminal ERASE and KILL characters.

~m *mesg-list* Reads the named messages into the message buffer, shifted right one TAB. If no messages are specified, reads the current message.

~M *mesg-list* Reads the named messages into the message buffer, shifted right one TAB. If no messages are specified, reads the current message.

~p Prints out the messages collected so far, prefaced by the message header fields.

~Print Prints the real names or descriptions (in parentheses) after each recipient in a header field.

~q Aborts the message being sent, copying the message to the dead.letter file in your home directory if **save** is set.

~r *filename* Reads the named file into the message buffer.

~Return *name* Adds the given names to the Return-receipt-to field.

~s *string* Causes the named *string* to become the current subject field.

~t *name...* Adds the given names to the direct recipient list.

~v Invokes a visual editor (defined by the VISUAL option) on the message buffer. After you quit the editor, you may resume appending text to the end of your message.

~w *filename* Writes the body of the message to the named file.

Options are controlled with the **set** and **unset** commands. An option may be either a switch, in which case it is either on or off, or a string, in which case the actual value is of interest. The switch options include the following:

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.

asksubject Causes **mail** to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field is sent.

autombox Causes all examined messages to be saved ir the user mailbox unless deleted or saved. autoprint Causes the **delete** command to behave like **dp** - thus, after deleting a message, the next one will be typed automatically.

chron	Causes messages to be displayed in chronological order.
dot	Permits use of dot (.) as the end-of-file character when composing messages.
ignore	Causes interrupt signals from your terminal to be ignored and echoed as at-signs (@).
mchron	Causes messages to be listed in numerical order (most recently received first), but displayed in chronological order.
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	Prevents aborted messages from being appended to the dead.letter file in your home directory on receipt of two interrupts (or a ~q).
quiet	Suppresses the printing of the version header when first invoked.
verify	Causes each target mail recipient to be verified. This option permits errors made while composing messages to be corrected or ignored.

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 the text editor to use in the **visual** command and ~v escape.

escape — If defined, the first character of this option gives the character to use in the place of the tilde (~) to denotes escapes.

page=*n* — Specifies the number of lines (*n*) to be printed in a "page" of text when displaying messages.

record — If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not saved.

toplines — If defined, gives the number of lines of a message to be printed out with the top command; normally, the first six lines are printed.

Files

/usr/spool/mail/*	System mailboxes
/usr/name/dead.letter	File where undeliverable mail is deposited.
/usr/name/mbox	Your old mail
/usr/name/.mailrc	File giving initial mail commands
/usr/lib/mail/aliases	System-wide aliases
/usr/lib/mail/aliases.hash	System-wide alias database
/usr/lib/mail/faliases	Forwarding aliases for the local machine
/usr/lib/mail/mailiases	Machine aliases
/usr/lib/mailhelp.cmd	Help file
/usr/lib/mailhelp.esc	Help file
/usr/lib/mailhelp.set	Help file
/usr/lib/mail/mailrc	System initialization file
/usr/bin/mail	The mail command

See Also

aliases(M), aliashash(M), netutil(C)

Notes

This utility was developed at the University of California at Berkeley and is used with permission.

If you use the C-shell to send mail, be sure to escape any exclamation point (!) used on the command line.

Name

make - Maintains, updates, and regenerates groups of programs.

Syntax

make [**-f** *makefile*] [**-p**] [**-i**] [**-k**] [**-s**] [**-r**] [**-n**] [**-b**] [**-e**] [**-u**] [**-t**] [**-q**] [*names*]

Description

The **make** command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

-f *makefile*
Description file name. *Makefile* is assumed to be the name of a description file.

-p Print out the complete set of macro definitions and target descriptions.

-i Ignore error codes returned by invoked commands. This mode is entered if the fake target name .IGNORE appears in the description file.

-k Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.

-s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.

-r Do not use the built-in rules.

-n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

-b Compatibility mode for old makefiles.

-e Environment variables override assignments within makefiles.

-u Force an unconditional update.

-t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

-q Question. The **make** command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

.DEFAULT
If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.

.PRECIOUS
Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT
Same effect as the **-s** option.

.IGNORE
Same effect as the **-i** option.

Make executes commands in *makefile* to update one or more target names. *Name* is typically a program. If no **-f** option is present, **makefile, Makefile,** and the Source Code Control System (SCCS) files **s.makefile,** and **s.Makefile** are tried in order. If *makefile* is **-**, the standard input is taken. More than one **-f** *makefile* argument pair may appear.

Make updates a target only if its dependents are newer than the target (unless the **-u** option is used to force an unconditional update). All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that

begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash> <new-line> sequence. Everything printed by **make** (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce:

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files a.o and b.o, and that they in turn depend on their corresponding source files (a.c and b.c) and a common file incl.h:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The SHELL environment variable can be used to specify which shell **make** should use to execute commands. The default is /bin/sh. The first one or two characters in a command can be the following: -, @, -@, or @-. If @ is present, printing of the command is suppressed. If - is present, **make** ignores an error. A line is printed when it is executed unless the -s option is present, or the entry .SILENT: is in *makefile*, or unless the initial character sequence contains a @. The -n option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed (see discussion of the MAKEFLAGS macro under "Environment"). The -t (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate **make.** If the **-i** option is present, or the entry .IGNORE: appears in *makefile,* or the initial character sequence of the command contains -. the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old makefiles (those written for the old version of **make**) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name .PRECIOUS.

Environment

The environment is read by **make.** All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The MAKEFLAGS environment variable is processed by **make** as containing any legal input option (except **-f** and **-p**) defined for the command line. Further, upon invocation, **make** "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, MAKEFLAGS always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **-n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would have been executed. This is because the **-n** is put in MAKEFLAGS and passed to further invocations of $(MAKE). This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string *include* appears as the first seven letters of a line in a *makefile*, and is followed by a blank or a tab, the rest of the line is assumed to be a file name and will be read by the current invocation, after substituting for any macros.

Macros

Entries of the form *string1 = string2* are macro definitions. *string2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under "Libraries."

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$* The macro $* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@ The $@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$< The $< macro is only evaluated for inference rules or the .DEFAULT rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the $< macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc -c -O $*.c

or:

.c.o:
        cc -c -O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form lib(file.o). In this case, **$** evaluates to **lib** and **$%** evaluates to the library member, file.o.

Four of the five macros can have alternative forms. When an upper case D or F is appended to any of the four macros, the meaning is changed to "directory part" for D and "file part" for F. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, **make** has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .f .f~ .sh .sh~
.c.o .c.a .c~.o .c~.c .c~.a
.f.o .f.a .f~.o .f~.f .f~.a
.h~.h .s.o .s~.o .s~.s .s~.a .sh~.sh
.l.o .l.c .l~.o .l~.l .l~.c
.y.o .y.c .y~.o .y~.y .y~.c
```

The internal rules for **make** are contained in the source file **rules.c** for the **make** program. These rules can be locally modified. To print out the rules compiled into the **make** on any machine in a form suitable for recompilation, the following command is used:

make -fp - 2>/dev/null </dev/null

A tilde in the above rules refers to an SCCS file (see **sccsfile**(F)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with **make**'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x*.**c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for .SUFFIXES. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h
           .h~ .f .f~
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; .SUFFIXES: with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because **make** has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, CFLAGS, LFLAGS, and YFLAGS are used for compiler options to **cc**(CP), **lex**(CP), and **yacc**(CP), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus lib(file.o) and $(LIB)(file.o) both refer to an archive library which contains file.o. (This assumes the LIB macro has been previously defined.) The expression $(LIB)(file1.o file2.o) is not legal. Rules pertaining to archive libraries have the form *.XX.***a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have lib(file.o) depend upon file.o explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
          $(AR)  $(ARFLAGS) $@ $*.o
          rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into **make** and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
      $(CC) -c $(CFLAGS) $(?:.o=.c)
      $(AR) $(ARFLAGS) lib $?
      rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

Files

[Mm]akefile and s.[Mm]akefile /bin/sh

See Also

cd(C), sh(C), and cc(CP), lex(CP), yacc(CP), printf(S), sccsfile(F) in the *Reference (CP, S, F)*

Notes

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

File names with the characters = : @ will not work. Commands that are directly executed by the shell, notably **cd**(C), are ineffectual across newlines in **make**. The syntax (lib(file1.o file2.o file3.o) is illegal. You cannot build lib(file.o) from file.o. The macro $(a:.o=.c~) does not work. Named pipes are not handled well.

Name

mesg - Allows or disallows messages sent to a terminal.

Syntax

mesg [-n] [-y]

Description

Mesg allows or disallows messages sent to a terminal and reports the current state without changing it. The arguments are:

-n Forbids messages via **write**(C) by revoking non-user write permission on the user's terminal.

-y Reinstates write permission.

Files

/dev/tty*

See Also

write(C)

Diagnostics

The exit status is 0 if messages are receivable, 1 if they are not, 2 if there is an error.

Name

mkdir - Makes a new directory.

Syntax

mkdir [**-m** *mode*] [**-p**] *dirname* ...

Description

The **mkdir** command creates a new directory. The standard entries "dot" (.), for the directory itself, and "dot dot" (..), for its parent, are made automatically.

A directory name can be up to 14 characters long. Do not use spaces or any of the following characters in the name:

```
*   ?   [   .
'   !   ]   "
;   (   \
:   )   ,
```

You can create several directories at one time by separating the directory names with spaces.

To use the **mkdir** command, you must have write permission in the current directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Options

The following options apply to **mkdir:**

-m Specifies the mode to be used for new directories. (See **chmod**(C) for types of modes.) The default is 777, modified by the **umask** value.

-p Creates *dirname* by creating all the non-existing parent directories first.

Examples

This command creates directories called Accounting, Engineering, and Marketing in the current directory.

mkdir Accounting Engineering Marketing Retn

To create the subdirectory structure ltr/jd/scott, type:

mkdir -p ltr/jd/scott

See Also

chmod(C), rmdir(C), umask(C)

Diagnostics

Mkdir returns exit code 0 if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero.

Name

mknod - Builds a special file.

Syntax

/etc/mknod *name* [**b**] | [**c**] *major minor*
/etc/mknod *name* **m**
/etc/mknod *name* **p**
/etc/mknod *name* **s**

Description

This command can only be used by the super user.

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name of the* file. The second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the major device type and the minor device (for example, unit, drive, or line number). They may be either decimal or octal.

The assignment of major device numbers is specific to each system. These numbers come from the system source file master.

Options

m Creates named shared data (memory)

p Creates named pipes

s Creates semaphores

Examples

The following command creates a block-type device named hd0d on major device number 0, minor device number 4.

/etc/mknod hd0d b 0 4 Retn

The next command creates a character device named tty03 on major device 10, tty number 3.

/etc/mknod /dev/tty03 c 10 3

See Also

mknod(S) in the *Reference (CP, S, F)*

Notes

If **mknod** is used to create a device in a remote directory (RFS), the major and minor device numbers are interpreted by the server.

Name

mkstr - Creates an error message file from 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 specified *file*, by placing a modified version of the input file in a file whose name consists of the specified *prefix* and the original name. The optional dash (-) causes the error messages to be placed at the end of the specified message file for recompiling part of a large **mkstr** program.

A typical **mkstr** command line is:

mkstr pistrings xx *.c

This command causes 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 newline character; the null character terminates the message so it can be easily used when retrieved, the newline character makes it possible to sensibly **cat**(C) the error message file to see its contents. The massaged copy of the input file then contains an **lseek**(S) pointer into the file which can be used to retrieve the message.

For example, the command changes the following:

```
error("Error on reading", a2, a3, a4);
```

into:

```
error(m, a2, a3, a4);
```

where *m* is the seek position of the string in the resulting error message file. The programmer must create a routine error which opens the message file, reads the string, and prints it out. The following example illustrates such a routine.

Example

```
char    efilname[] = "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(C);
                }
        }

        if (lseek(efil, (long) a1, 0) || read(efil,
                 buf, 256) <= 0)
           goto oops;
        printf(buf, a2, a3, a4);
}
```

See Also

lseek(S), xstr(CP)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Name

more - Views a file one screen at a time.

Syntax

more -[cdflrsuw] [*-n*] [*+linenumber*] [*+/pattern*] [*file ...*]

Description

Using the **more** command, you can examine continuous text one screen at a time. **More** pauses at the end of each screen, printing:

--More--

If you type Retn at the "--More--" message, one more line is displayed. If you press the Spacebar, another full screen of the text is displayed.

If **more** is reading from a file, rather than a pipe, the "--More--" message also contains a percentage telling you the amount you have read so far.

More looks in the environment variable MORE to preset any flags desired. For example, if you prefer to view files using the **-c** option, the shell command **MORE=-c** in the .profile file causes all invocations of **more** to use this mode.

Options

-c Redraws each full screenful of text from the top of the screen instead of scrolling. When you use **-c**, the *-n* option is ignored. The **-c** option is ignored if the terminal does not have the ability to clear to the end of the line.

-d Displays the message "Hit space to continue, Del to abort" at the end of each screen. This is useful if **more** is being used as a filter.

-f Causes **more** to count logical, rather than screen lines. That is, long lines are not folded.

-l Causes **more** to ignore form feed control characters (Ctrl-L). If this option is not given, **more** pauses after any line that contains a Ctrl-L, as if the end of a screen had been reached. Also, if a file begins with a form feed, the screen is cleared before the file is printed.

-r Causes any control characters that **more** does not recognize to be displayed as ^*x*, where Ctrl-x is the unrecognized control character.

-s Removes multiple blank lines from the output. This is especially helpful when viewing **nroff** output, maximizing the useful information present on the screen.

-u Suppresses underlining in the source file.

-w Causes **more** to prompt you with the message:

"--No more--"

and wait for you to press any key before exiting. Without this option, you automatically exit to system level at the end of the file.

-n Displays *n* lines in the specified file.

+linenumber Starts the file display at the specified line number.

+/pattern Starts the file display two lines before the line containing the pattern.

There are several responses you can use when **more** pauses at the "--More--" prompt. With the exception of *i/expr*, you don't need to press Retn after these options.

i Ctrl-d	Displays 11 more lines. If you specify the *i* option, scroll size is changed from 11 to what you specify.
d	Same as Ctrl-d.
:f	Displays the current file name and line number.
h or **?**	Describes all the **more** commands.
i Space	Displays *i* more lines, or another screenful if no argument is given.
i/*expr*	Searches 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, 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.
*i***f**	Skips *i* screenfuls and prints a screen full of lines.
*i***:n**	Skips to the *i*th next file given in the command line (skips to the last file if *i* doesn't make sense).
*i***:p**	Skips to the *i*th previous file given in the command line. If in the middle of printing out a file, **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 rings.
*i***s**	Skips *i* lines and prints a screenful of lines.
q or **Q**	Exits from **more**.

q or **:Q** Same as **q** or **Q**.

v Starts up the screen editor **vi**(C) at the current line.

= Displays the current line number

. Repeats the previous command. What you type will not show on your screen, except for the slash (/) and exclamation (!) commands.

' Goes 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* Invokes a shell with the specified 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.

Up to the time when the command character itself is given, you can press the line kill character to cancel the numerical argument being formed. In addition, you can press the erase character to redisplay the "--More--(*nn*%)" message.

Examples

This command displays the file **pretzels** one screen at a time.

more pretzels

This command displays a file named **marzipan,** 15 lines at a time, beginning two lines before the expression "munchkin."

more -15 +/munchkin marzipan

This command pipes the nroff output from the file **jiminy** through the **more** command so you can preview the **nroff** output.

nroff -mm jiminy|more +2 -s

where:

-mm	is a macro package referenced by **nroff**
more	pipes the **nroff** output through the **more** command
-s	removes blank lines from the output
+2	begins at line number 2

Files

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

Related Commands

cat(C), sh(C)

See Also

environ(M)

Name

mount, umount - Mounts and unmounts file systems and remote resources.

Syntax

/etc/mount [[**-r -c**] [**-f** *fstyp*] *special directory*]
/etc/mount [[**-r -c**] [**-d**] *resource directory*]
/etc/umount *special*
/etc/umount [**-d**] *resource*

Description

File systems other than root (/) are considered removable in the sense that they can be either available to users or unavailable. **Mount** announces to the system that *special*, a block special device or *resource*, a remote resource, is available to users from the **mount** point *directory*. *Directory* must exist already; it becomes the name of the root of the newly mounted *special*.

Mount, when entered with arguments, adds an entry to the table of mounted devices, /etc/mnttab. **Umount** removes th entry. If invoked with no arguments, **mount** prints the entire **mount** table. If invoked with an incomplete argument list, **mount** searches /etc/fstab for the missing arguments.

The following options are available:

-r Indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected, this flag must be used.

-d Indicates that *resource* is a remote resource that is to be mounted on *directory* or unmounted. To mount a remote resource Remote File Sharing must be up and runnin and the resource must be advertised by a remote computer. If **-d** is not used, *special* must be a local block device.

-c Does not use RFS client caching.

-f *fstyp*	Indicates that *fstyp* is the file system type to be mounted. If this argument is omitted, it defaults to the root *fstyp*.
special	Indicates the block special device that is to be mounted on *directory*.
resource	Indicates the remote resource name that is to be mounted on a directory.
directory	Indicates the directory mount point for *special* or *resource*. (The directory must already exist.)

Umount announces to the system that the file system previously mounted *special* or *resource* is to be made unavailable. If invoked with an incomplete argument list, **umount** searches /etc/fstab for the missing arguments.

Mount can be used by any user to list mounted file systems and resources. Only the super-user can mount and unmount file systems.

Files

/etc/mnttab	Mount table
/etc/fstab	File system table

See Also

fuser(M), setmnt(C), and mount(S), umount(S), fstab(F), mnttab(F) in the *Reference (CP, S, F)*.

adv, rfstart, unadv in the *Remote File Sharing* manual

Diagnostics

If the **mount**(S) system call fails, **mount** prints an appropriate diagnostic. **Mount** issues a warning if the file system to be mounted is currently mounted under another name. A remote resource mount will fail if the resource is not available or if Remote File Sharing is not running.

The error message "mount: Object is remote" occurs when an attempt is made to mount local devices or remote resources on a directory that is remote or has a remote resource mounted on it.

Umount fails if *special* or *resource* is not mounted or if it is busy. *Special* or *resource* is busy if it contains an open file or some user's working directory. In such a case, you can use **fuser**(M) to list and kill processes that are using *special* or *resource.*

Notes

Physically removing a mounted file system diskette from the diskette drive before issuing the **umount** command damages the file system.

Name

mountall, umountall - Mounts, unmounts multiple file systems.

Syntax

/etc/mountall [-] [*file-system-table*] ...
/etc/umountall [-k]

Description

These commands may be executed only by the super-user.

Mountall is used to mount file systems according to a *file-system-table* (/etc/fstab is the default file system table). The special file name "-" reads from the standard input.

Before each file system is mounted, it is checked using **fsstat**(M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using **fsck**(M), before the mount is attempted.

Umountall causes all mounted file systems except **root** to be unmounted. The **-k** option sends a SIGKILL signal, via **fuser**(M), to processes that have files open.

Files

File-system-table format:

column 1	block special file name of file system
column 2	mount-point directory
column 3	"-r" if to be mounted read-only; "-d" if remote
column 4	(optional) file system type string
column 5+	ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

```
/dev/dsk/c1d0s2 /usr -r S51K
```

See Also

fsck(C), fsstat(M), fstab(M), fuser(M), mount(C), sysadm(C), and signal(S), in the *Reference (CP, S, F)*

Diagnostics

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from **fsck**(C), **fsstat**(M), and **mount**(C).

Name

multiuser, singleuser - Causes the system to enter multi-user or single-user mode.

Syntax

/etc/multiuser
/etc/singleuser

Description

This command can only be used by the super-user.

Multiuser changes the system mode of operation from single-user to multi-user. **Multiuser** performs system startup functions such as mounting file systems and starting various daemons and spoolers. The **/etc/telinit 2** command is executed to tell **init**(M) to enter multi-user mode (run level 2).

Singleuser causes the system to kill all currently running processes and enter system maintenance mode (run level 1).

See Also

init(M), shutdown(C), who(C)

Name

mv - Moves or renames files and directories.

Syntax

mv [**-f**] *file1 file2*
mv [**-f**] *file... directory*
mv [**-f**] *directory1 directory2*

Description

Mv moves (changes the name of) *file1* to *file2.*

If *file2* already exists, it is removed before *file1* is moved. If *file2* is write-protected, **mv** prints the mode (see the **chmod**(C) command) and reads the standard input to obtain a line; if the line begins with y, the move takes place; if not, **mv** exits.

If you use the **-f** (force) option, the move takes place regardless of *file2*'s mode.

If you move one or more files to a specified directory, the files retain their original file names.

Mv refuses to move a file onto itself.

If *file1* and *file2* are in different file systems, **mv** must copy the file and delete the original. The owner name becomes that of the copying process and any linking relationship with other files is lost.

Directories may not be moved across filesystems.

Examples

The following command changes the name of the existing file rhubarb to the new file name alfalfa.

mv rhubarb alfalfa

This command moves the files alfalfa, alfredo, and alfresco to the directory /usr/al.

mv alfalfa alfredo alfresco /usr/al

The following command moves the directory seeds/flowers to the directory seeds/mums.

mv seeds/flowers seeds/mums

Files

/usr/lib/mv_dir	Program to move directories

See Also

cp(C), chmod(C)

Name

nawk - Pattern scanning and processing language.

Syntax

nawk [**-F** *re*] [*parameter...*] ['*prog*'] [**-f** *progfile*] [*file...*]

Description

Nawk is a pattern scanner and language processor. The latest version of **nawk** provides capabilities unavailable in previous versions, including new built-in functions and variables, and the ability to use 8-bit character sets. The previous version of **nawk** is called **awk**(C).

The **-F** *re* option defines the input field separator to be the regular expression *re*.

Parameters, in the form *x=... y=...* may be passed to **nawk**, where *x* and *y* are **nawk** built-in variables (variables are discussed later).

Nawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (') to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *file* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the **-f** *progfile* option.

Input files are read in order; if there are no files, the standard input is read. The file name - means the standard input. Each input 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 normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the **-F** *re* option.) The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the form:

pattern { *action* }

Either *pattern* or *action* may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

expression relop expression
expression matchop regular expression

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

var **in** *array*

or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line has been read and after the last input line has been read, respectively.

Regular expressions are as in **egrep**(C). In patterns they must be surrounded by slashes. 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 pattern.

A regular expression may be used to separate fields by using the -F *re* option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

ARGC	Command line argument count
ARGV	Command line argument array
FILENAME	Name of the current input file
FNR	Ordinal number of the current record in the current file
FS	Input field separator regular expression (default blank)
NF	Number of fields in the current record (default 0)
NR	Ordinal number of the line of the current record
OFMT	Output format for numbers (default %.6g)
OFS	Output field separator (default blank)
ORS	Output record separator (default new-line)
RS	Input record separator (default new-line)

An action is a sequence of statements. A statement may b one of the following:

if (*conditional*) *statement* [**else** *statement*]
while (*conditional*) *statement*
do *statement* **while** (*conditional*)
for (*expression* ; *conditional* ; *expression*) *statement*
for (*var* **in** *array*) *statement*
delete *array*[*subscript*]
break
continue
{ [*statement*] ... }
expression # commonly variable = expression
print [*expression-list*] [>*expression*]
printf *format* [, *expression-list*] [>*expression*]
next # skip remaining patterns on this input line
exit [*expr*] # skip the rest of the input; exit status
is expr
return [*expr*]

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole input 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 or zero. 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 *>expression* is present, or on a pipe if | *cmd* is present. The arguments are separated by the current output field separator and terminated by the output record separator. The **printf** statement formats its expression list according to a format string (see **printf**(S) in the *Reference (CP, S, F)*).

Nawk has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: **atan2, cos, exp, int, log, rand, sin, sqrt,** and **srand.** **Int** truncates its argument to an integer. **Rand** returns a random number between 0 and 1. **Srand** (*expr*) sets the seed value for **rand** to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

gsub(*for, repl, in*)
Behaves like **sub** (see below), except that it replaces successive occurrences of the regular expression (like the **ed**(C) global substitute command).

index(*s, t*) Returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.

length(*s*) Returns the length of its argument taken as a string, or of the whole line if there is no argument.

match(s, *re*) Returns the position in string s where the regular expression *re* occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.

split(s, *a*, *fs*)
Splits the string s into array elements *a*[*1*], *a*[*2*], *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator FS if *fs* is not given.

sprintf(*fmt*, *expr*, *expr*,...)
Formats the expressions according to the **printf**(S) format given by *fmt* and returns the resulting string.

sub(*for*, *repl*, *in*)
Substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, **nawk** substitutes in the current record (**$0**).

substr(s, *m*, *n*)
Returns the *n*-character substring of s that begins at position *m*.

The input/output and general functions are:

close(*filename*)
Closes the file or pipe named *filename*.

cmd|**getline** Pipes the output of *cmd* into **getline**; each successive call to **getline** returns the next line of output from *cmd*.

getline Sets $0 to the next input record from the current input file.

getline <*file* Sets $0 to the next record from *file*.

getline *var* Sets variable *var* instead.

getline *var* <*file*
Sets *var* from the next record of *file*.

system(*cmd*) Executes *cmd* and returns its exit status.

All forms of **getline** return 1 for successful input, 0 for end of file, and -1 for an error.

Nawk also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

function *name*(*args*,...) { *stmts* }
func *name*(*args*,...) { *stmts* }

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

Examples

The following examples are taken from a file.

Print lines longer than 72 characters:

```
'length > 72'
```

Print the first two fields in opposite order:

```
'{ print $2, $1 }'
```

Same as above, with input fields separated by comma and/or blanks and tabs:

```
'BEGIN { FS = ",[ \t]*[ \t]+" }
       { print $2, $1 }'
```

Add up the first column, print the 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 }'
```

Simulate **echo**(C):

```
'BEGIN {
        for (i = 1; i < ARGC; i++)
                printf "%s", ARGV[i]
        printf "\n"
        exit
        }'
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
       { print }
```

command line: **nawk -f program n=5 input**

See Also

grep(C), awk(C), sed(C), lex(C)
printf(S) in the *Programmer's Guide*

Notes

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

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

Name

newgrp - Logs user in to a new group.

Syntax

newgrp [-] [*group*]

Description

Newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by **newgrp**, regardless of whether it terminated successfully or due to an error condition (e.g., unknown group).

Exported variables retain their values after invoking **newgrp**; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, suppose a user has a primary prompt string (PS1) other than $ (default) and has not exported PS1. After an invocation of **newgrp**, successful or not, their PS1 will now be set to the default prompt string **$**. Note that the shell command **export** (see **sh**(C)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, **newgrp** changes the group identification back to the group specified in the user's password file entry. This is a way to undo the effect of an earlier **newgrp** command.

If the first argument to **newgrp** is a -, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in /etc/group as being a member of that group.

Files

/etc/group	System's group file
/etc/passwd	System's password file

See Also

login(C), sh(C), environ(M) group(M), passwd(M)

Notes

There is no convenient way to enter a password into /etc/group. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

Name

nice - Runs a command at a different priority.

Syntax

nice [*-increment*] *command* [*arguments*]

Description

Nice executes *command* with a lower CPU scheduling priority. **Nice** values range from 0 to 39, where 0 yields the highest priority and 39 the lowest. By default, commands have a value of 20.

If an *-increment* argument is given, where *increment* is in the range 1-19, increment is added to the default nice of 20 to produce a numerically higher value, meaning a lower scheduling priority.

If no increment is given, an increment of 10 to produce a value of 30 is assumed.

The super-user may run commands with a higher priority than normal by using a double negative increment. For example, an argument of "--10" would decrement the default to produce a value of 10, increasing the scheduling priority.

See Also

nice(S), nohup(C)

Diagnostics

Nice returns the exit status of the subject command.

Notes

If you specify a value outside the range, a value greater than 39 is equivalent to 39; a value less than zero is equivalent to zero.

Name

nl - Adds line numbers to a file.

Syntax

nl [-b*type*] [-h*type*] [-f*type*] [-v*start #*] [-i*incr*] [**-p**] [-l*num*] [-s*sep*] [-w*width*] [-n*format*] [-d*delim*] [*file*]

Description

Nl reads lines from the named *file*, or from the standard input if no file is named, and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are available independently for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines in the body).

The start of logical page sections is signaled by input lines containing nothing but the following character(s):

Page Section	Line Contents
Header	\:\:\:
Body	\:\:
Footer	\:

Unless signaled otherwise, **nl** assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional filename. Only one file may be named. The options are:

-b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:

a	number all lines
t	number lines with printable text only (the default)
n	no line numbering
p *string*	number only lines that contain the regular expression specified in *string*

-h*type* Same as -b*type* except for header. The default type for logical page header is **n** (no lines numbered).

-f*type* Same as -b*type* except for footer. The default for logical page footer is n (no lines numbered).

-p Does not restart numbering at logical page delimiters.

-v*start#* *Start#* is the initial value used to number logical page lines. The default is 1.

-i*incr* *Incr* is the increment value used to number logical page lines. The default is 1.

-s*sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default is TAB.

-w*width* *Width* is the number of characters to be used for the line number. The default width is 6.

-n*format* *Format* is the line numbering format. The recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. The default format is **rn** (right justified).

-l*num* *Num* is the number of blank lines to be considered as one. For example, -l2 results in only the second adjacent blank being numbered (if the appropriate -ha, -ba, and/or -fa option is set). The default is 1.

-d*xx* You can change the delimiter characters, *xx*, specifying the start of a logical page section, from the default characters (\:) to two user-specified characters. If you enter only one character, the second character remains the default character (:). Do not enter a space between **-d** and the delimiter characters. To enter a backslash (\) use two backslashes (\\).

Example

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

See Also

num(C), pr(C)

Name

nohup - Runs a command immune to hangups and quits.

Syntax

nohup *command* [*arguments*]

Description

Nohup executes *command* with hangups and quits ignored. If output is not redirected by the user, it will be sent to nohup.out. If nohup.out is not writable in the current directory, output is redirected to $HOME/nohup.out.

Examples

It is frequently desirable to apply **nohup** to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. You can then issue:

nohup sh *file*

and the **nohup** applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type **sh** can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ingnored (see **sh**(C)):

nohup *file* **&**

An example of what the contents of *file* could be is:

sort *ofile* > *nfile*

See Also

nice(C), sh(C), signal(S)

Notes

The command:

nohup *command1;command2*

nohup applies only to *command1*. To correct this, use:

nohup (*command1;command2*)

Name

num - Numbers lines.

Syntax

num [*file...*]

Description

The lines in the specified *files*, or the standard input, are copied to the standard output preceded by line numbers. Tabs remain aligned in the ouput as the lines are printed preceded by the number, blank padded to six digits, and then two spaces.

See Also

see(C), nl(C)

(BLANK)

Name

od - Displays files in octal format.

Syntax

od [**-bcdhosx**] [*file*] [[**+**] *offset* [**.**][**b**]]

Description

Od displays *file* in one or more formats as selected by the first argument (**-o** is default). The format options are:

-b Interprets bytes in octal.

-c Interprets bytes in ASCII. Certain nongraphic 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 Interprets words in decimal.

-h Interprets words in hexadecimal (same as **-x**).

-o Interprets words in octal.

-s Interprets words in signed decimal.

-x Interprets words in hexadecimal (same as **-h**).

The *file* argument specifies which file is to be displayed. If no file argument is specified, the standard input is used.

The *offset* argument (normally interpreted as octal bytes) specifies the offset in the file where the display is to start. If **.** is appended, the *offset* is interpreted in decimal. If **b** is appended, the *offset* is interpreted in 512-byte blocks. If *file* is omitted, *offset* must be preceded by **+**. The display continues until EOF.

See Also

adb(C), hd(C)

(BLANK)

Name

pack, pcat, unpack - Compresses and expands files.

Syntax

pack [-] [**-f**] *name ...*
pcat *name ...*
unpack *name ...*

Description

Pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If **pack** is successful, *name* will be removed. Packed files can be restored to their original form using **unpack** or **pcat**.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress. No packing will occur if:

- The file appears to be already packed
- The filename has more than 12 characters
- The file has links
- The file is a directory
- The file cannot be opened
- No disk storage blocks will be saved by packing
- A file called *name*.z already exists
- The .z file cannot be created
- An I/O error occurred during processing

The last segment of the filename must not contain more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what **cat**(C) does for ordinary files. The specified files are unpacked and written to the standard output. Thus, to view a packed file named name.z use:

pcat name.z

or just:

pcat name

To make an unpacked copy, say nnn, of a packed file, name.z (without destroying name.z), use the command:

pcat name >nnn

Pcat returns the number of files it was unable to unpack. Failure may occur if:

- The filename (exclusive of the .z) has more than 12 characters

- The file cannot be opened
- The file does not appear to be the output of **pack**

Unpack expands files created by **pack.** For each *name* file specified in the command, a search is made for a file called *name*.z (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in **pcat**, as well as in a file where the "unpacked" name already exists, or if the unpacked file cannot be created.

See Also

cat(C), passwd(M)

Name

passwd - Changes login password.

Syntax

passwd [*name*]

Description

This command changes or installs a password associated with the login name.

Ordinary users may change only the password which corresponds to their login name.

Passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered, **passwd** checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and **passwd** terminates; see **passwd**(F). Note that password aging is NOT enabled by default. See **passwd**(M) for more information.

Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant.
- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means upper and lower-case letters.

- Each password must differ from the user's login name and any reverse or circular shift of that login name. For comparison purposes, an upper case letter and its corresponding lower-case letter are equivalent.

- New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower-case letter are equivalent.

One whose effective user ID is zero is called a super-user; see **id**(C) and **su**(C). Super-users may change any password; hence, **passwd** does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

Files

/etc/passwd

See Also

login(C), id(M), su(M) and crypt(S), passwd(F) in the *Reference (CP, S, F)*

Name

pconfig - Port configuration utility.

Syntax

pconfig [**-f** | **-h** | **-i**]

Description

The **pconfig** command defines port configuration information for your system; with **pconfig**, you can set up ports on your computer as terminal, modem, or printer devices. You can also set parameters, such as communication speed (baud rate) and terminal type for these devices.

You must be the super-user to use this command.

Options

The options are as follows:

-f Forces execution even if someone else is using **pconfig**

-h Displays a brief help introduction

-i Initializes modem ports

After you invoke the **pconfig** command, the prompt at the bottom of the screen asks you to type one of the following commands:

a Adds a new port specification

c Changes a port assignment

d Deletes a port assignment

q Updates the system port assignment files, then terminates the program

r Configures a remote printer.

^W Displays help information

! Passes a command to the operating system for execution

Each of these command options is described below. To select an option, just type the character. You do not need to press Retn. When executing one of the options, you may, however, be asked to enter a word. In this case, follow the word with a Retn. In most cases, entering just a Retn will leave that particular command or attribute unchanged.

a Add a port. The named port will be added to the port configuration list, and eventually to the /etc/inittab file.

c Change a port. You will be asked to specify the port to change. Enter the name of the port as it appears when you use the **d** option to display the port assignments. You will be asked to enter the device type: terminal, printer, or other.

Select **terminal**, and you will be asked to specify the terminal, then the action for the port. "Respawn" means the port is enabled, "off" means disabled. Finally, you will be asked for the baud rate (communication speed) of the port. The baud rate is determined by the protocol defined in the /etc/gettydefs file, and is selected by an identifier that is in the first field of each line in /etc/gettydefs. Enter the desired identifier to set the baud rate.

To set the port up as a modem, select the identifier that contains the word "MODEM." For a bidirectional line (dial out and dial in), select the identifier that contains "UUCP."

Select **printer**, and you will be asked to enter the name of the printer, which can be **lp** or **lp***NN*, where *NN* is a digit between 0 and 255. Enter the baud rate as described above.

Select **other** only if you want to directly edit the fields within the /etc/inittab file. These fields will then be written to the /etc/inittab file without error checking.

q Quits the program. If any changes have been made to the port assignments, you will be asked if you want to save the changes. Type **y** to install the new configuration, or **n** to exit the program without making any changes.

! Escapes to a sub-shell invoked from within **pconfig.** If the SHELL environment variable is initialized, the specified shell will be run; if not, /bin/sh is invoked.

During the execution of the quit command, the /etc/inittab file is copied to /etc/oinittab. Thus, if a problem occurs during the use of this program, you can recover the prior state of the port configurations.

Files

/etc/inittab
/etc/gettydefs
/etc/printers
/usr/lib/PCF

See Also

init(M), inittab(M), getty(M), gettydefs(M)
Operations Guide

Name

pg - File perusal filter for CRTs.

Syntax

pg [*-number*] [**-p** *string*] [**-cefns**] [*+linenumber*] [*+/pattern/*] [*files...*]

Description

The **pg** command is a filter that allows the examination of *files* one screenful at a time on a CRT. (The file name dash (-) and/or NULL arguments indicate that **pg** should read from the standard input.) Each screenful is followed by a prompt. If you type a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, **pg** scans the **terminfo**(M) data base for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal type "dumb" is assumed.

The command line options are:

-number An integer specifying the size (in lines) of the window that **pg** is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23.)

-p *string* Causes **pg** to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

-c Moves the cursor to home and clears the screen before displaying each page. This option is ignored if clear_screen is not defined for this terminal type in the **terminfo**(M) data base.

-e Causes **pg** *not* to pause at the end of each file.

-f Normally, **pg** splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The **-f** option inhibits **pg** from splitting lines.

-n Normally, commands must be terminated by a Newline character. This option causes an automatic end of command as soon as a command letter is entered.

-s Causes **pg** to print all messages and prompts in standout mode (usually inverse video).

+linenumber
Start up at *linenumber*.

+/pattern/
Start up at the first line containing the regular expression pattern.

The responses that may be typed when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands that cause further perusal normally take a preceding address, an optionally signed number indicating the point from which further text should be displayed. This address is interpreted in either pages or lines depending on the command. A signed address specifies a point relative to the current page or line, and an unsigned address specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<Newline> or <Blank>
This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes **pg** to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no address.

. or **^L** Causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in **ed**(C) are available. They must always be terminated by a <Newline>, even if the **-n** option is specified.

i/pattern/
Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?
Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, **pg** will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

You can modify the environment of perusal with the following commands:

*i*n Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i*p Begin perusing the *i*th previous file in the command line. The *i* is an unsigned number, default is 1.

*i*w Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the s and *filename* is optional. This command must always be terminated by a <Newline>, even if the -n option is specified.

h Help by displaying an abbreviated summary of available commands.

q or Q Quit pg.

!*command* *Command* is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a <Newline>, even if the -n option is specified.

At any time when output is being sent to the terminal, you can press the quit key (normally Ctrl-\) or the interrupt (Break) key. This causes pg to stop sending output, and display the prompt. You may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then pg acts just like cat(C), except that a header is printed before each file (if there is more than one).

Example

A sample usage of **pg** in reading system news would be:

```
news | pg -p"(Page%d):"
```

Notes

While waiting for terminal input, **pg** responds to Break, Del, and ^ by terminating execution. Between prompts, however, these signals interrupt **pg**'s current task and place you in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the **z** and **f** commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

Files

/usr/lib/terminfo/?/*	Terminal information database
/tmp/pg*	Temporary file when input is from a pipe

See Also

ed(C), grep(C), terminfo(M)

Bugs

If terminal tabs are not set every eight positions, undesirable results may occur.

When using **pg** as a filter with another command that changes the terminal I/O options, terminal settings may not be restored correctly.

Name

pr - Prints files.

Syntax

pr [[*-column*] [**-w***width*] [**-a**]] [**-e***ck*] [**-i***ck*] [**-drtfp**] [**+***page*] [**-n***ck*] [**-o***offset*] [**-l***length*] [**-s***separator*] [**-h** *header*] [*file...*]

pr [[**-m**] [**-w***width*]] [**-e***ck*] [**-i***ck*] [**-drtfp**] [**+***page*] [**-n***ck*] [**-o***offset*] [**-l***length*] [**-s***separator*] [**-h** *header*] *file1 file2...*

Description

Pr is used to format and print the contents of a file. If *file* is -, or if no files are specified, **pr** assumes standard input. **Pr** prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with **-h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multi-column output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the *separator* character.

Either *-column* or **-m** should be used to produce multi-column output. **-a** should only be used with *-column* and not **-m**.

Command line options are:

+page	Begin printing with page numbered *page* (default is 1).
-column	Print *column* columns of output (default is 1). Output appears as if **-e** and **-i** are turned on for multi-column output. May not use with **-m**.
-a	Print multi-column output across the page one line per column. *Columns* must be greater than one. If a line is too long to fit in a column, it is truncated.
-m	Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with *-column*.
-d	Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.
-e*ck*	Expand input tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
-i*ck*	In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

-n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of -m output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

-w*width* Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (*-column* and -m). There is no line limit for single column output.

-o*offset* Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the *width* and *offset*.

-l*length* Set the length of a page to *length* lines (default is 66). -l0 is reset to -l66. When the value of *length* is 10 or less, -t appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When -l*length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text.

-h *header* Use *header* as the text line of the header to be printed instead of the file name. -h is ignored when -t is specified or -l*length* is specified and the value of *length* is 10 or less. (-h is the only **pr** option requiring space between the option and argument.

-p Pause before beginning each page if the output is directed to a terminal (**pr** will ring the bell at the terminal and wait for a carriage return).

-f Use single form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

-r Print no diagnostic reports on files that will not open.

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **-t** overrides the **-h** option.

-s*separator* Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless **-w** is specified.

Examples

Print *file1* and *file2* as a double-spaced, three-column listing headed by "file list":

pr -3dh "file list" *file1 file2*

Copy *file1* to *file2*, expanding tabs to columns 10, 19, 28, 37,... :

pr -e9 -t <*file1* >*file2*

Print *file1* and *file2* simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

pr -t -n *file1* | **pr -t -m -n** *file2* -

Files

/dev/tty*	To delay messages enabling them to print at the bottom of files rather than interspersed throughout printed output.

See Also

cat(C), pg(C)

Name

printenv - Prints out the environment.

Syntax

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; otherwise, it returns status 0.

See Also

sh(C), environ(M), csh(C)

Name

ps - Reports process status.

Syntax

ps [**-adefl**] [**-c** *corefile*] [**-s** *swapdev*] [**-t** *tlist*] [**-p** *plist*] [**-u** *ulist*] [**-g** *glist*]

Description

Ps prints certain information about active processes. If you use the **ps** command without arguments, it lists information about processes associated with the current terminal. If you use arguments with the **ps** command, more specialized information is listed.

Options

-a Prints information about all processes, except process group leaders and processes not associated with a terminal.

-c *corefile* Uses the file *corefile* in place of /dev/mem.

-d Prints information about all processes, except process group leaders.

-e Prints information about all processes.

-f Generates a full listing. Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed. See below for a description of each column in a full listing.

Under the **-f** option, **ps** tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **-f** option, is printed in square brackets.

-g *glist* Lists information about processes whose process groups are given in *glist*. *Glist* is a list of process group leaders and is in the same format as *tlist*.

-l Generates a long listing. See below.

-p *plist* Lists information about processes whose process ID numbers are given in *plist*. *Plist* is in the same format as *tlist*.

-s *swapdev* Uses the file *swapdev* in place of /dev/swap. This is useful when examining a corefile.

-t *tlist* Lists information about the processes associated with the terminals given in *tlist*. *Tlist* can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

-u *ulist* Lists information about processes whose user ID numbers or login names are given in *ulist*. *Ulist* is in the same format as *tlist*. In the listing, the numerical user ID is printed unless the **-f** option is used, in which case the login name is printed.

When you type a **ps** command, the status of all processes running on your system is displayed in columns across your screen. The meaning of each column in a **ps** listing is given below, as well as the options that cause that column to appear (**-l** for the long option and **-f** for the full option). When **all** is listed as the option, it means that the column always displays no matter what option you enter.

Column Heading	Option	Description
F	l	A status word consisting of flags associated with the process. Each flag is associated with a bit in the status word. These flags are ored to form a single hex number. Process flag bits and their meanings are: 0x00001 System (resident) process. 0x00002 Process is being traced. 0x00004 Stopped process given to parent by wait system call. 0x00008 Process cannot wakeup by a signal. 0x00010 In core. 0x00020 Process cannot be swapped. 0x00040 Set when signal goes remote. 0x00080 Process in stream poll. 0x00100 Process is being stopped via /proc. 0x00200 Signal tracing via /proc. 0x00400 Doing I/O via /proc. 0x00800 Stop on exec. 0x01000 Process is open via /proc. 0x02000 U-block is in core. 0x04000 Set process to run on last /proc close. 0x08000 Process asleep. 0x10000 Processing exiting via ptrace. 0x20000 Process stopped within a call to sleep. 0x40000 U-block is being swapped in or out. 0x80000 Waiting for u-block swap to complete.

Column Heading	Option	Description
S	l	The state of the process 0 non-existent S sleeping W waiting R runable I intermediate Z terminated T stopped X waiting for memory P running
UID	f,l	The user ID number of the process owner; the login name is printed under the -f option.
PID	all	The process ID of the process; it is possible to kill a process if you know this data.
PPID	f,l	The process ID of the parent process.
C	f,l	Process utilization for scheduling, in hex.
STIME	f	Starting time of the process.
PRI	l	The priority of the process; higher numbers mean lower priority.
P	l	CPU on which process last run.
NI	l	Nice value; used in priority computation.
ADDR	l	The memory address of the process, if resident; otherwise, the disk address.
SZ	l	The virtual size in 1K units of the stack and data regions process.

Column Heading	Option	Description
WCHAN	l	The event for which the process is waiting or sleeping; if blank, the process is runable, in hex.
TTY	all	The controlling terminal for the process. If using windows, also shows number of the window in which process is running.
TIME	all	The cumulative executive time for the process.
CMD	all	The command name; the full command name and its arguments are printed under the **-f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked <defunct>.

Examples

This command lists your processes using the long form.

```
ps -l
```

```
 F S  UID  PID  PPID  C PRI NI ADDR SZ WCHAN STIME    TTY  TIME CMD
 1 S  109 1605     1  0  30 20   57 24  aac8 09:07:12 3/2  0:06  sh
 1 R  109 5856  1605 23  61 20   63 32       09:30:20 4/0  0:03  ps
```

Files

/dev/mem	Memory
/dev	Searched to find swap device and terminal ("tty") names
/dev/swaptab	Contains kernel structure list of all swap devices

Related Commands

kill(C)

Notes

If **ps** can't find the swap device, you will see an error message:

Cannot open *swapdev*

where *swapdev* is a swap device such as /dev/hd0a.

Name

pscreen - Sets up terminal to enable print-screen capability.

Syntax

pscreen -k *key-seq* [**-e** *escape-seq*] [**-d** *device*]
[-t *terminal type*]

Description

The **pscreen** command sets up the terminal so that when a specified key is pressed on the keyboard, the contents of the screen will be printed on a printer connected to the auxiliary port on the terminal. **Pscreen** does not directly cause the screen to be printed on the printer; its only function is to set things up so that whenever the specified key is pressed, the screen will be printed.

The terminal that is being used must have an auxiliary port on it and it must accept an escape sequence that will cause the contents of the screen to be printed on a printer connected to the auxiliary port.

Some terminals such as the Altos V have a PRINT key that will automatically print the screen. If you are using such a terminal, it is not necessary to use **pscreen.** However, other terminals such as the Altos II and Altos III do not have such a key, but they will accept an escape sequence to print the screen. On such terminals, it is suggested that a function key be reserved for the print screen key and that **pscreen** be used to define the sequence of characters that the function key sends to the computer when the key is pressed.

Options

-k Specifies the *sequence of characters* that the terminal sends to the computer when the print-screen key is pressed. The **-k** option must be specified.

-e Specifies the *escape sequence* that the computer should send to the terminal after the print-screen key is pressed. If not specified, then the string of characters specified in the termcap capability "SP" is used.

-d Specifies the *device* name (e.g., /dev/tty04 or tty04). If not specified, the user's current port is used. Only the super-user can specify a device other than the current port.

-t Specifies the terminal type (e.g., altos3). The terminal type must be one of the terminal types in the /etc/termcap file.

If neither -t nor -d are specified, then the terminal type specified in the environment variable TERM is used. If -t is not specified but -d is specified, then the terminal type specified in /etc/ttytype for the specified device is used.

Each character in the sequences specified by the -k and -e options can be either a single ASCII character, a hexadecimal number (e.g., 0x1b), an octal number (e.g., 033), or an ASCII abbreviation (e.g., ESC or SOH). Alternatively, the character can be specified as a decimal number provided that it contains at least 2 digits. A single decimal digit (0 through 9) will be interpreted as a single ASCII character unless a trailing "d" (or "D") is appended (e.g., 9D). The character must have a non-zero value less than 0x80 (i.e., less than 128).

The following list shows all the valid ASCII abbreviations. Each abbreviation can be either uppercase or lowercase letters.

Abbreviation	Hex Value	Abbreviation	Hex Value
SOH	0x01	DLE	0x10
STX	0x02	DC1	0x11
ETX	0x03	DC2	0x12
EOT	0x04	DC3	0x13
ENQ	0x05	DC4	0x14
ACK	0x06	NAK	0x15
BEL	0x07	SYN	0x16
BS	0x08	ETB	0x17
HT	0x09	CAN	0x18
LF	0x0a	EM	0x19
VT	0x0b	SUB	0x1a
FF	0x0c	ESC	0x1b
CR	0x0d	FS	0x1c
SO	0x0e	GS	0x1d
SI	0x0f	RS	0x1e
		US	0x1f

Examples

pscreen -k SOH j CR -e ESC [0i -t altos3

and

pscreen -k 0x01 0x6a 0x0d -e 0x1b 0x5b 0x69 -t altos3

Both of these commands are identical. The shifted F11 key on the Altos III produces the "SOH j CR" sequence. So when that key is pressed, the screen will be printed on the printer.

Notes

Be careful about using the special shell characters in the command line for **pscreen**. For example, when using the C shell (csh), the above example should be typed as follows:

pscreen -k SOH j CR -e ESC '[' i -t altos3

because the left bracket is a special character.

If the printer is also configured as an auxiliary (transparent) printer using **pconfig**, it is not possible to print the screen while the printer is busy printing another file.

Name

pwd - Prints the working directory name.

Syntax

pwd

Description

The **pwd** command prints the pathname of the working (current) directory. Using **pwd**, you can always check to see where you are in the system.

Example

The system responds that you are on machine x1 in the WorkNet network in the directory /usr/chris. If your computer is not part of a network, the system response would be "/usr/chris" with no machine name.

```
$ pwd

@x1/usr/chris
```

See Also

cd(C)

Diagnostics

"Cannot open ..." and "Read error in ..." indicate possible file system trouble; contact your system administrator.

Name

quot - Summarizes file system ownership.

Syntax

quot [*option*] ... [*filesystem*]

Description

Quot prints the number of blocks in the named *filesystem* currently owned by each user. If no *filesystem* is named, the file systems given in /etc/mnttab are examined.

The following *options* are available:

-c Prints three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file. Data for files of size greater than 499 blocks are included in the figures for files of exactly size 499.

-f Prints count of number of files as well as space owned by each user.

-n This option uses the output of the **ncheck**(C) command to produce a list of files and their owners, in the specified filesystem. For example:

ncheck /dev/hd0b | sort +0n | quot -n /dev/hd0b

will produce a listing of all files and their owners, on the root file system.

Files

/etc/passwd	Gets user names
/etc/mnttab	Contains list of mounted file systems

See Also

du(C), ls(C)

Notes

Holes in files are counted as if they actually occupied space.

Name

random - Generates a random number.

Syntax

random [**-s**] [*scale*]

Description

Random generates a random number on the standard output, and returns the number as its exit value. By default, this number is either 0 or 1. If *scale* is given a value between 1 and 255, then the range of the random value is from 0 to *scale*. If *scale* is greater than 255 an error message is printed.

-s Silent: returns the random number as an exit value but is not printed on the standard output. If an error occurs, **random** returns an exit value of zero.

See Also

rand(S)

Notes

This command does not perform any floating point computations. **Random** uses the time of day as a seed.

Name

reboot - Automatically reboots the system.

Syntax

/etc/reboot

Description

The **reboot** command, when used with the **haltsys** command, automatically reboots the system.

To use **reboot** with the **haltsys** command, enter:

/etc/reboot Retn
/etc/haltsys Retn

Related Commands

haltsys(C), shutdown(M)

Name

recover - Restores the contents of a file system from streaming tape to disk.

Syntax

recover [-i|-s|-v] *mag_tape file_system*

Description

This command can only be used by the super-user.

The **recover** command copies the tape (specified by *mag_tape*) to the hard disk file system. Specify *file_system* as /dev/rhd1b for the second hard disk, or /dev/rhd2b for the third hard disk.

Recover can only restore a tape that was backed up using the **archive**(C) command. To restore a tape created with the **dump.hd**(C) command, use **restore.hd**(C).

Options

-i Displays the character string that was specified by the **-i** option on the **archive** command that created the tape. This option does not copy the contents of the tape to the hard disk; it only displays the character string.

-s Displays information from the header block: the number of the tape, the creation date, the starting block number on the tape, and the name of the file system. This option does not copy the contents of the tape to the hard disk.

-v Verifies the checksums on the tape (makes sure the data was written correctly), without writing to the hard disk.

When restoring the root file system on the hard disk (i.e., /dev/rhd0b), boot the system from the Root floppy disk, and select option "c" on the menu. The tape used by this procedure must have been created by the **dump.hd**(C)

command, or by the backup commands of the AOM menu system or Business shell.

Be sure to specify /dev/rsct (for streaming tape) when using **archive** or **recover.** Do not use /dev/rct, because the tape will "stream" only with /dev/rsct.

Examples

For example, this command backs up the second hard disk to tape.

/etc/umount /dev/hd1b Retn
archive -e /dev/rhd1b /dev/rsct Retn

This command restores the files on the archive tape to the second hard disk.

/etc/umount /dev/hd1b Retn
recover /dev/rsct /dev/rhd1b Retn

To see the name of the device you want to back up, use:

mount Retn

The screen displays the device name, for example,

```
/usr2 on /dev/hd1b ...
```

Related Commands

archive(C)

See Also

Operations Guide

Name

reset - Resets the teletype bits to soft-copy terminal standard mode.

Syntax

reset

Description

The **reset** command sets the teletype bits to a sensible state with the erase character set to Ctrl-h and the kill character set to Ctrl-x. The **reset** command is most useful when a program that you are running in raw mode (terminal input is passed to a program one character at a time) fails.

To reset your terminal using this command, you may have to type **reset** followed by a linefeed (or Ctrl-j if there is no linefeed).

Related Commands

stty(C)

Name

restore.hd - Restores a hard disk from tape.

Syntax

restore.hd

Description

The **restore.hd** command restores the root file system from cartridge tape (made with **dump.hd**(C)) to the hard disk. To run **restore.hd**, boot the system from the Root floppy disk, then select option "C," Restore data to the hard disk from cartridge tape. To restore files from the second (third) hard disk, see **recover**(C).

CAUTION

Restore.hd overwrites ALL data on the hard disk and replaces it with the files from the cartridge tape.

See Also

archive(C), layout(C), recover(C), sizefs(C)
Operations Guide

Name

rev - Reverses lines of a file.

Syntax

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.

Notes

There is a limit of 255 characters per line.

Name

rm, rmdir - Removes files or directories.

Syntax

rm [-f] [-i] *file...*
rm -r [-f] [-i] *dirname...* [*file...*]
rmdir [-p] [-s] *dirname...*

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, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **-f** option is in effect.

Rmdir removes the named directories, which must be empty.

Three options apply to **rm**:

-f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

-r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory con-

tains. The write-protected files are removed without prompting, however, if the **-f** option is used, or if the standard input is not a terminal and the **-i** option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the **-f** option is used), resulting in an error message.

-i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **-f** option and remains in effect even if the standard input is not a terminal.

Two options apply to **rmdir:**

-p This option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

-s This option is used to suppress the message printed on standard error when **-p** is in effect.

Diagnostics

All messages are generally self-explanatory. It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

```
rm -r .*
```

Both **rm** and **rmdir** return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

See Also

unlink(S), rmdir(S) in the *Reference (CP, S, F)*

Name

rmail - Receives mail (from uucp link).

Synopsis

rmail *to-path*
text

Description

This front-end mailer is only for systems with "execmail." It takes the first lines of a message from the standard input, and folds "From" lines to produce a single line with an accurate **uucp**(C) path, and pipes the mail through usr/lib/mail/execmail *from-path to-path.*

Text is the text of the letter on the standard input until an Ctrl-d (end-of-file).

See Also

uucp(C)

Name

sar - System activity reporter.

Syntax

sar [**-aAbcCdDmpqrSuvwy**] [**-o** *file*] **t** [*n*]
sar [**-aAbcCdDmpqrSuvwy**] [**-s** *time*] [**-e** *time*] [**-i** *sec*] [**-f** *file*]

Description

Sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If the **-o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, sar extracts data from a previously recorded *file*, either the one specified by **-f** option or, by default, the standard system activity daily data file /usr/adm/sa/sa*dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **-s** and **-e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **-i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

-a Report use of file access system routines: iget/s, namei/s, dirblk/s.

-A Report all data. Equivalent to **-aAbcCdDmpqrSuvwy**

-b Report buffer activity:

bread/s, bwrit/s	transfers per second of data between system buffers and disk or other block devices
lread/s, lwrit/s	accesses of system buffers
%rcache, %wcache	cache hit ratios, i.e.,

%rcache, %wcache	cache hit ratios, i.e., (1-bread/lread) as a percentage
pread/s, pwrit/s	transfers via raw (physical) device mechanism
pread/s, pwrit/s	transfers via raw (physical) device mechanism. When used with -D, buffer caching is reported for locally-mounted remote resources

-c Report system calls:

scall/s	system calls of all types
sread/s, swrit/s, fork/s, exec/s	specific system calls
rchar/s, wchar/s	characters transferred by read and write system calls

When used with -D, the system calls are split into incoming, outgoing, and strictly local calls.

-C Report Remote File Sharing buffer caching overhead:

snd-inv/s	number of invalidation messages per second sent by your machine as a server
snd-msg/s	total outgoing RFS messages sent per second
rcv-inv/s	number of invalidation messages received from the remote server
rcv-msg/s	total number of incoming RFS messages received per second
dis-bread/s	number of buffer reads that woul be eligible for caching if caching were not turned off (indicates the penalty of running unchached)

blk-inv/s — number of buffers removed from the client cache.

-d Report activity for each block device, e.g., disk or tape drive. When data is displayed, the device specification *dsk*is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent.

The activity data reported is:

%busy, avque — portion of time device was busy servicing a transfer request, average number of requests outstanding during that time

r+w/s, blks/s — number of data transfers from or to device, number of bytes transferred in 512-byte units

avwait, avserv — average time in milliseconds that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times)

-D Report Remote File Sharing activity. When used in combination with **-u** or **-c**, it causes **sar** to produce the remote file sharing version of the corresponding report (**-u** is assumed when neither **-u** or **-c** is specified).

-m Report message and semaphore activities:

msg/s, sema/s — primitives per second

-p Report paging activities:

vflt/s — address translation page faults (valid page not in memory)

pflt/s — page faults from protection errors (illegal access to page) or "copy-on-writes"

pgfil/s — vflt/s satisfied by page-in from file system

rclm/s — valid pages reclaimed for free list

-q Report average queue length while occupied, and % of time occupied:

runq-sz, %runocc — run queue of processes in memory and runnable

swpq-sz, %swpocc — swap queue of processes swapped out but ready to run

-r Report unused memory pages and disk blocks:

freemem — average pages available to user processes

freeswap — disk blocks available for process swapping

-S Report server and request queue status: average number of Remote File Sharing servers on the system (serv/lo- hi), % of time receive descriptors are on the request queue (request %busy), average number of receive descriptors waiting for service when queue is occupied (request avg lgth), % of time there are idle servers (server %avail), average number of idle servers when idle ones exist (server avg avail).

-u Report CPU utilization (the default):

%usr, %sys, %wio, %idle — portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle. When used with -D, %sys is split into percent of time servicing requests from remote machines (%sys remote) and all other system time (%sys local)

-v Report status of process, i-node, file tables:

text-sz, proc-sz, inod-sz, file-sz, lock-sz
entries/size for each table, evaluated once at sampling point; ov - overflows that occur between sampling points for each table

-w Report system swapping and switching activity:

swpin/s, swpot/s, bswin/s, bswot/s
number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs)

pswch/s process switches

-y Report TTY device activity:

rawch/s, canch/s, outch/s
input character rate, input character rate processed by canon, output character rate

rcvin/s, xmtin/s, mdmin/s
receive, transmit and modem interrupt rates.

Examples

To see today's CPU activity so far:

sar

To watch CPU activity evolve for 10 minutes and save data:

sar -o temp 60 10

To later review disk and tape activity from that period:

sar -d -f temp

Files

/usr/adm/sa/sa*dd*	Daily data file, where *dd* are digits representing the day of the month.

See Also

sar(M)

Name

script - Makes a record of your terminal session.

Syntax

script [**-a**] [**-q**] [**-S** *shell*] [*file*]

Description

Script makes a file of everything printed on your terminal. The typescript is saved in a file, and can be sent to the line printer later with **lpr**(C). If a *file* name is given, the typescript is saved there. If not, the typescript is saved in the file named typescript.

To exit **script**, type Ctrl-d. This sends an end-of-file to all processes you have started up, and causes **script** to exit. For this reason, Ctrl-d behaves as though you had typed an infinite number of them.

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 Causes **script** to append to the typescript file instead of creating a new file.

-q Asks for "quiet mode", where the "script started" and "script done" messages are turned off.

-S Lets you specify the shell; the default depends on the system. If the variable SHELL is set in the environment, it is used if possible.

Notes

Since the operating system has no way to write an end of file down a pipe without closing the pipe, there is no way to simulate a single Ctrl-d without ending **script**.

The new shell has its standard input coming from a pipe rather than a tty, so **stty**(C) will not work, and neither will **ttyname**. In particular, this means that screen editors such as **vi**(C) are inoperative.

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

sdb - Symbolic debugger.

Syntax

sdb [-w] [-W] [*objfile* [*corfile* [*directory-list*]]]

Description

The **sdb** command calls a symbolic debugger that can be used with C programs. It may be used to examine object files and core files and to provide a controlled environment for program execution.

Objfile is an executable program file which has been compiled with the **-g** (debug) option. If it has not been compiled with the **-g** option, the symbolic capabilities of **sdb** will be limited, but the file can still be examined and the program debugged. The default for *objfile* is **a.out.** *Corfile* is assumed to be a core image file produced after executing *objfile;* the default for *corfile* is **core.** The core file need not be present. A - in place of *corfile* will force **sdb** to ignore any core image file. The colon-separated list of directories *(directory-list)* is used to locate the source files used to build *objfile.*

It is useful to know that at any time there is a *current line* and *current file.* If *corfile* exists then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main*(). The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfile* cannot be found, or are newer than *objfile.* This checking feature and the accompanying warnings may be disabled by the use of the **-W** flag.

Names of variables are written just as they are in C. **Sdb** does not truncate names. Variables local to a procedure may be accessed using the form *procedure:variable.* If no procedure name is given, the procedure containing the current line is used by default.

You can also refer to structure members as *variable.member*, pointers to structure members as *variable->member*, and array elements as *variable*[*number*]. Pointers may be dereferenced by using the form *pointer*[0]. Combinations of these forms may also be used. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by **sdb**. An unqualified structure variable may also be used with various commands. Generally, **sdb** will interpret a structure as a set of variables. Thus, **sdb** will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value **sdb** displays, not the addresses of individual elements.

Elements of a multi-dimensional array may be referenced as *variable [number][number]...*, or as *variable [number,number,...]*. In place of *number*, the form *number;number* may be used to indicate a range of values, * may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, **sdb** displays all the values of an array or of the section of an array if trailing subscripts are omitted.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants that are valid in C may be used, so that addresses may be input in decimal, octal, or hexadecimal.

Line numbers in the source program are referred to as *file-name:number* or *procedure:number*. In either case, the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used

by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under **sdb**, all addresses refer to the executing program; otherwise they refer to *objfile* or *corfile*. An initial argument of -w permits overwriting locations in *objfile*. In order to overwrite a location in *obfile*, the process must not be running and a corefile must be present.

Addresses

The offset in a file associated with a virtual 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 offset* corresponding to a virtual *address* is calculated as follows:

b1 <=*address* < *e1*

file address=*address*+*f1*-*b1*

b2 <=*address* < *e2*

file address=*address*+*f2*-*b2*,

or 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.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected 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.

In order for **sdb** to be used on large files, all appropriate values are kept as unsigned 32-bit integers.

Commands

The commands for examining data in the program are:

t Print a stack trace of the terminated or halted program.

T Print the top line of the stack trace.

variable/clm
Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

b one byte
h two bytes (half word)
l four bytes (long word)

Legal values for *m* are:

c character

d decimal

u decimal, unsigned

o octal

x hexadecimal

f 32-bit single precision floating point

g 64-bit double precision floating point

s Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.

a Print characters starting at the variable's address. This format may not be used with register variables.

p pointer to procedure

i Disassemble machine-language instruction with addresses printed numerically and symbolically.

I Disassemble machine-language instruction with addresses just printed numerically.

Length specifiers are only effective with the **c**, **d**, **u**, **o** and **x** formats. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, **sdb** chooses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells **sdb** to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count specifier is used for the **s** or **a** command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command **./**.

The **sh**(C) metacharacters * and ? may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

linenumber?*lm*
variable:?*lm*

Print the value at the address from **a.out** or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

variable=lm
linenumber=lm
number=lm
Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then **lx** is used. The last variant of this command provides a convenient way to convert between decimal, octal, and hexadecimal.

variable!value
Set *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted *'character*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type *double*. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

x Print the machine registers and the current machine-language instruction.

X Print the current machine-language instruction.

The commands for examining source files are:

e *procedure*
e *filename*
e *directory/*
e *directory filename*

The first two forms set the current file to the file containing *procedure* or to *filename*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, filename, or directory is given, the current procedure name and filename are reported.

/regular expression/
Search forward from the current line for a line containing a string matching *regular expression* as in **ed**(C). The trailing / may be deleted.

?regular expression?
Search backward from the current line for a line containing a string matching *regular expression* as in **ed**(C). The trailing ? may be deleted.

p Print the current line.

z Print the current line followed by the next 9 lines. Set the current line to the last line printed.

w Window. Print the 10 lines around the current line.

number
Set the current line to the given line number. Print the new current line.

count+
Advance the current line by *count* lines. Print the new current line.

count-
Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

count **r** *args*
count **R**
Run the program with the given arguments. The **r** command with no arguments reuses the previous arguments to the program while the **R** command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber **c** *count*
linenumber **C** *count*
Continue after a breakpoint or interrupt. If *count* is given, the program will stop when *count* breakpoints have been encountered. The signal which caused the program to stop is reactivated with the **C** command and ignored with the **c** command. If a line number is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

linenumber **g** *count*
Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

s *count*
S *count*
Single step the program through *count* lines. If no count is given then the program is run for one line. **S** is equivalent to **s** except it steps through procedure calls.

i
I Single step by one machine-language instruction. The signal which caused the program to stop is reactivated with the **I** command and ignored with the **i** command.

variable\$**m** *count*
address:**m** *count*
Single step (as with **s**) until the specified location is modified with a new value or *count* instructions have been executed. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

level **v**
Toggle verbose mode, for use when single stepping with **S**, **s** or **m**. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

k Kill the program being debugged.

procedure(arg1,arg2,...)
procedure(arg1,arg2,...)/m
Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to **d**. This facility is only available if the program was loaded with the **-g** option.

linenumber **b** *commands*
Set a breakpoint at the given line. If a procedure name without a line number is given (e.g., "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the **-g** option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to **sdb**. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If **k** is used as a command to execute at a breakpoint, control returns to **sdb**, instead of continuing execution.

B Print a list of the currently active breakpoints.

linenumber **d**
Delete a breakpoint at the given line. If no *linenumber* is given then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d** then the breakpoint is deleted.

D Delete all breakpoints.

l Print the last executed line.

linenumber **a**
Announce. If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber* **b l**. If *linenumber* is of the form *proc:*, the command effectively does a *proc:* **b T**.

Miscellaneous commands:

!*command*
The command is interpreted by **sh**(C).

newline
If the previous command printed a source line, then advance the current line by one line and print the new current line. If the previous command displayed a memory location, then display the next memory location.

end-of-file character
Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last. The end-of-file character is usually control-d.

< *filename*
Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When **sdb** is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

M Print the address maps.

M [?/] [*]*b e f*
Record new values for the address map. The argument? and / specify the text and data maps, respectively. The first segment (*b1, e1, f1*) is changed unless * is specified, in which case the second segment (*b2, e2, f2*) of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

" *string*
Print the given string. The C escape sequences of the form *character* are recognized, where *character* is a nonnumeric character.

q Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

V Print the version number.

Q Print a list of procedures and files being debugged.

Y Toggle debug output.

Files

a.out
core

See Also

cc(C), a.out(F), core(F), sh(C)

Notes

When **sdb** prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The size is assumed to be **int** (integer).

Data which are stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

Name

sdiff - Compares files side-by-side.

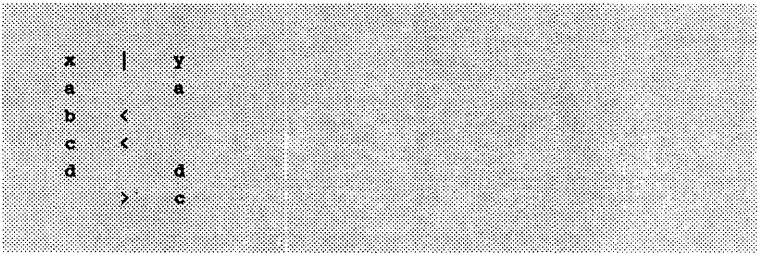
Syntax

sdiff [*options*] *file1 file2*

Description

Sdiff uses the output of **diff**(C) to produce a side-by-side listing of two files indicating the lines that are different. The lines of the two files are printed with a blank gutter between them if they are identical; a < is put in the gutter if the line only exists in *file1*, a > is put in the gutter if the line only exists in *file2*, and a | is put in the gutter for lines that are different.

For example:



The following options exist:

-l Only prints the left side of any lines that are identical.

-o *output* Uses the next argument, *output*, as the name of a third file created by merging *file1* with *file2*. Identical lines of *file1* and *file2* are copied to output. Sets of differences, as produced by **diff**(C), are printed; sets of differences share a common gutter character. After printing each set of differences, **sdiff** prompts the user with a % and waits for one of the following commands:

e Calls the editor with a zero length file

e l Calls the editor with the left column

e r Calls the editor with the right column

e b Calls the editor with the concatenation of left and right

l Appends the left column to the output file

q Exits from the program

r Appends the right column to the output file

s Turns on silent mode; does not print identical lines

v Turns off silent mode

-s Does not print identical lines.

-w *n* Uses the next argument, *n*, as the width of the output line. The default line length is 130 characters.

Upon exiting from the editor, the resulting file is concatenated onto the end of the output file.

See Also

diff(C), ed(C)

Name

sed - Invokes a stream editor.

Syntax

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** options, 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.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval. 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, i.e., a */regular expression/* in the style of **ed**(C) modified thus:

- In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef.**
- The escape sequence **\n** matches a new-line *embedded* the pattern space.

- A period . matches any character except the *terminal* newline of 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).

In the following list of functions, the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. 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. The *rfile* or *wfile* argument 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 new-line. 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 new-line. (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 new-line 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**(C). *Flags* is zero or more of:

n n = 1 - 512. Substitute for just the *n*th occurrence of the regular expression.

g Global. Substitute for all non-overlapping 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.

(0)# If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

See Also

awk(C), ed(C), grep(C)

Name

see - Displays a file

Syntax

see [-] [*file...*]

Description

See lists a file, displaying non-printing characters in visible format. Control characters show as "^X" for Ctrl-x, where *x* is any letter. Tab prints as "^I." Delete prints as "^?." Ends of lines are marked with "$" unless the "-" option is given.

See Also

cat(C), ex(C)

Name

setmnt - Establishes /etc/mnttab table.

Syntax

/etc/setmnt

Description

The **setmnt** command is usually executed by the system and creates and updates the /etc/mnttab table, which is needed for both the **mount**(C) and **umount**(C) commands. **Setmnt** reads the standard input and creates a mnttab entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's special file (for example, /dev/hd0b) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the **mnttab** entry.

Files

/etc/mnttab

See Also

mount(C)

Notes

Problems may occur if *filesys* or *node* are longer than 32 characters. **Setmnt** silently enforces an upper limit on the maximum number of **mnttab** entries.

Name

setmode - Port modes utility.

Syntax

setmode *device mode ...*

Description

Setmode sets tty modes (see **tty**(M)) for tty ports that are used for serial devices. Use this program to set baud rate, tab expansion, and newline actions for programs that communicated directly through a serial port.

Setmode takes a list of tty *mode*s from its command line, does an **stty**(C) on the indicated device, and sleeps forever, which keeps the *device* open with the desired modes. Invoke **setmode** once for each port device.

To ensure that **setmode** is run every time the system enters multi-user mode, invoke **setmode** in the **/etc/inittab** file.

You must invoke **setmode** with at least two arguments: the name of the *device* (special file) and at least one tty *mode*.

Files

/dev/tty*	tty devices
/etc/inittab	

Related Commands

disable(C), enable(C), pconfig(C), stty(C), xtty(C), inittab(M)

See Also

tty(M)

Name

setmodem - Sets and unsets a tty port to be used with a modem.

Syntax

/etc/setmodem *mode* **tty***nn*

Description

This command can only be accessed by the super user.

Use the **setmodem** command to set up a device (/dev/tty*nn*) for use with a modem. The letters *nn* stand for a one or two-digit device number, for example, **tty05.** Execute this command every time the system is booted for every port that has a modem attached.

The **setmodem** command ensures that a dial-up tty will be logged out when a telephone connection is terminated.

Options

Mode is either **on, off,** or **user.**

on Sets **clocal** to OFF, and **hupcl** to ON. This flag cannot be changed without issuing another **setmodem** command.

off Sets **clocal** to ON, and **hupcl** to OFF. This flag cannot be changed without issuing another **setmodem** command.

user is as follows:

If the **clocal** flag is not set, a high-to-low signal on pin 6 causes a hang up; a low-to-high signal allows login to occur.

If the **hupcl** flag is set, the system sends a hangup signal when the last file connected to that terminal is closed.

You can change modes with the **stty**(C) command.

Examples

These commands are equivalent and tell the system that a modem is being used on serial port 5.

```
/etc/setmodem on /dev/tty05
/etc/setmodem on tty05
```

Related Commands

disable(C), tty(C), enable(C), getty(M), login(C), pconfig(C)

Name

setpgrp - Executes a command in a new process group.

Syntax

setpgrp *command* [*arg ...*]

Description

This command creates a new process group to execute the specified command. It also removes the controlling tty from the new process group. **Setpgrp** can be useful for detaching commands that are run in the background from the parent shell process.

Diagnostics

Setpgrp returns an exit code of 1 if the command cannot be executed. Otherwise, the exit code is that returned by the command.

See Also

exec(S), setpgrp(S) in the *Reference (CP, S, F)*

Name

settime - Changes the access and modification dates of files.

Syntax

settime [*mmddhhmm*] [*yy*] [**-f** *sfile*] *file* ...

Description

This command sets the access and modification dates for one or more files. The dates are set to the specified date.

-f Sets *file* to the access and modification dates of *sfile*.

Use one of these methods to specify the new date. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last two digits of the year and is optional. For example:

settime 1008004586 ralph pete

sets the access and modification dates of files named ralph and pete to Oct 8, 12:45 AM, 1986. Another example:

settime -f ralph john

This sets the access and modification dates of the file named john to those of the file named ralph.

See Also

touch(C)

Name

sh, rsh - Shell, the standard/restricted command programming language.

Syntax

sh [**-acefhiknrstuvx**] [*args*]
rsh [**-acefhiknrstuvx**] [*args*]

Description

Sh is a command programming language that executes commands read from a terminal or a file. **Rsh** is a restricted version of the standard command interprepter **sh**; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See "Invocation" that follows for the meaning of arguments to the shell.

Definitions

A blank is a tab or a space. A name is a sequence of letters, digits, or underscores beginning with a letter or underscore. A parameter is a name, a digit, or any of the characters *****, **@**, **#**, **?**, **-**, **$**, and **!**.

Commands

A simple shell command is a sequence of 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. The value of a simple command is its exit status if it terminates normally, or (octal) 200+ status if it terminates abnormally, i.e., if the failure produces a core file.

A pipeline is a sequence of one or more commands separate(by a vertical bar (|). The caret (^) also has the same effect. The standard output of each command but the last is connected by a pipe 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 &. Of these symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does not wait for that pipeline to finish). The symbol && (||) causes the list following it to be executed only if the preceding pipeline returns a zero (nonzero) exit status. An arbitrary number of newlines may appear in a list, instead of semicolons, to delimit commands.

A command is either a simple-command or one of the following. Unless otherwise stated, 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* taken from the **in** *word* list. If **in** *word* is omitted, the **for** command executes the **do** list once for each positional parameter that is set (see "Parameter Substitution" below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [|*pattern*] ...) *list* ;;] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file name generation (see "File Name Generation" below) except that a slash, leading dot, or dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status; the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed. If its value is zero, the *list* following the next **then** is executed. Failing that, the **else**

list is executed. If no **else** *list* or **then** *list* is executed, the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the **list** is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status. You can use **until** in place of **while** to negate the loop termination test.

(*list*)

Executes *list* in a subshell.

{*list*;}

List is executed in the current (that is, parent) shell.

name () {*list*; }

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see "Execution"). The curly brace (}) must be on a line by itself, or preceded by a semicolon (;) or followed by a delimiter.

Use **type** *name* to display the commands executed by *name*.

The following words are only recognized as the first word of a command and when not quoted (not preceded by a bacl slash (\)):

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (``) can be used as part or all of a word; trailing newlines from the standard output are removed. No interpretation is done on the string before it is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes ("...`...`..."), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a new-line character (\new-line), both the backslash and the new-line are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslahses that precede characters other than \,`," , new-line, and $ are left intact when the command string is read.

Parameter Substitution

The character $ is used to introduce substitutable parameters. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=value [*name=value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same name.

${*parameter*}

A *parameter* is a sequence of letters, digits, or underscores (a name), a digit, or any of the characters *, @, #, ?, -, $, and !. 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. A name must begin with a letter or underscore. If *parameter* is a digit, it is a positional parameter. If *parameter* is * or @, all the positional parameters, starting with $1, are substituted (separated by spaces). *Parameter* **$0** is set from argument zero when the shell is invoked.

${*parameter*:-*word*}

If *parameter* is set and is non-null, substitute its value; otherwise, substitute *word*.

${*parameter*:=*word*}

If *parameter* is not set or is null, set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned in this way.

${*parameter*:?*word*}

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}

If *parameter* is set and is non-null, substitute *word*; otherwise, substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string. In the following example, **pwd** is executed only if **d** is not set or is null:

echo ${d:- `pwd`}

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (``) can be used as part or all of a word; trailing newlines from the standard output are removed. No interpretation is done on the string before it is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes ("...`...`..."), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a new-line character (\new-line), both the backslash and the new-line are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslahses that precede characters other than \,`," , new-line, and $ are left intact when the command string is read.

Parameter Substitution

The character $ is used to introduce substitutable parameters. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=value [*name=value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same name.

${*parameter*}

A *parameter* is a sequence of letters, digits, or underscores (a name), a digit, or any of the characters *, @, #, ?, -, $, and !. 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. A name must begin with a letter or underscore. If *parameter* is a digit, it is a positional parameter. If *parameter* is * or @, all the positional parameters, starting with $1, are substituted (separated by spaces). *Parameter* **$0** is set from argument zero when the shell is invoked.

${*parameter*:-*word*}

If *parameter* is set and is non-null, substitute its value; otherwise, substitute *word*.

${*parameter*:=*word*}

If *parameter* is not set or is null, set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned in this way.

${*parameter*:?*word*}

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}

If *parameter* is set and is non-null, substitute *word*; otherwise, substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string. In the following example, **pwd** is executed only if **d** is not set or is null:

echo ${d:- `pwd`}

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set.

Shell Parameters

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last synchronously executed command.

$ The process number of this shell.

! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the **cd** command.

PATH The search path for commands (see "Execution" below). The user may not change PATH if executing from **rsh.**

MAIL If this is set to the name of a mail file and MAILPATH is not set, then the shell informs the user of the arrival of mail in the specified file.

CDPATH The search path for the **cd** command.

MAILCHECK This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the MAILPATH or MAIL parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH — A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is "You have mail."

PS1 — Primary prompt string, by default "$".

PS2 — Secondary prompt string, by default ">".

IFS — Internal field separators, normally space, tab, and new-line.

SHACCT — If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. SHACCT must be exported for this to work.

SHELL — When the shell is invoked, it scans the enviroment (see "Environment" that follows) for this name. If it is found and 'rsh' is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to PATH, PS1, PS2, MAILCHECK and IFS, while HOME and MAIL are set by **login**(C).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in IFS). When such characters are found, they are split into distinct arguments. Explicit null arguments are retained. Implicit null arguments (those resulting from **parameters** that have no values) are removed.

Input/Output

Before a command is executed, its input and output can be redirected using a special notation interpreted by the shell. The following character strings may appear anywhere in a simple command or may precede or follow a command. These character strings 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 to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[--]*word* After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting word, or to an end-of-file. If, however, -- is appended to <<:

1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on word),

2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and

3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see "Quoting," later), no additional processing is done to the shell input. If no characters of *word* are quoted:

1) parameter and command substitution occurs,

2) (escaped) \new-line is ignored, and

3) \ must be used to quote the characters \, $, and `.

The resulting document becomes the standard input.

<&*digit* The standard input is duplicated from file descriptor *digit*. 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, 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, (a duplicate of file descriptor 1).

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced previously under "Commands," if a command is composed of several simple commands, redirection will be evaluated for the entire command before it is evaluated for each simple command. That is, the shell evaluates redirection for the entire list, then each pipeline within the list, then each command within each pipeline, then each list within each command.

If a command is followed by an ampersand (&), 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.

Redirection of output is not allowed in the restricted shell.

File Name Generation

Following substitution, each command word is scanned for the characters *, ?, and [. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically-sorted file names that match the *pattern*. If no file name is found that matches the *pattern*, the word is left unchanged. A period (.) at the start of a file name, or immediately following a slash (/), must be matched explicitly. (The slash (/) must be explicitly matched as well.)

These characters and their matching patterns are:

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted (preceded with a backslash (\)):

; & ( ) | ^ < > newline space tab

A character may be quoted (i.e., made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks (' ' or " "). During processing, the shell may quote certain characters to prevent

them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair \new-line is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, " ' ").

Inside a part of double quote marks (" "), parameter and command substitution occurs and the shell quotes the results to avoid blank intepretation and file name generation. If $* is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("$1 $2 ..."); however, if $@ is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("$1" "$2" ...). \ quotes the characters \, ', ", and $. The pair \new-line is removed before parameter and command substitution. If a backslash precedes characters other than \, ', ", $, and new-line, then the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of PS2) is is used.

Environment

The environment (see **environ**(M)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. 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 (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any simple command may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 *cmd args*

and

(export TERM; TERM=450; *cmd*)

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, all keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **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 an ampersand (&); otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11 (memory fault). (Also see the **trap** command under "Special Commands.")

Execution

Each time a command is executed, the above substitutions are carried out. If the command name does not match a Special Command, but matches the name of a defined function, the function is executed in the shell process (note

how this differs from the execution of shell procedures). The positional parameters $1, $2, ... are set to the arguments of the function. If the command name matches neither a Special Command nor the name of a defined function, a new process is created and an attempt is made to execute the command via **exec**(S).

The shell parameter PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is :/bin:/usr/bin (specifying the current directory, /bin, and /usr/bin, in that order). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a /, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an binary executable 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.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary execs later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the PATH variable is changed or the **hash -r** command is executed (see "Special Commands").

Special Commands

Input/output redirection is now permitted for these commands, although they cannot be used in pipelines. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Reads and executes commands from *file* and returns. The search path specified by PATH is used to find the directory containing *file*.

break [*n*]
Exits from the enclosing **for** or **while** loop, if any. If *n* is specified, then breaks *n* levels.

continue [*n*]
Resumes the next iteration of the enclosing **for** or **while** loop. If *n* is specified, then resumes at the *n*th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter HOME is the default *arg*. The shell parameter CDPATH defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a \ the search path is not used. Otherwise, each directory in the path is searched for *arg*. The **cd** command may not be executed by **rsh**.

echo [*arg* ...]
Echo arguments. See **echo**(C) for usage and description.

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 shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file will also cause the shell to exit).

export [*name* ...]
The given *names* are marked for automatic export to the environment of subsequently executed commands. If no arguments are given, a list of all names marked for export in this shell is printed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are not exported.

getopts Use in shell scripts to support command syntax standard (see **intro**(C)); it parses positional parameters and checks for legal options. See **getopts**(C) for usage and description.

hash [**-r**] [*name* ...]
For each name, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented, along with columns titled *hits* and *cost*. *Hits* is the number of times a command has been invoked by the shell.

Cost is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the hits information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* .... See **newgrp**(C) for usage and description.

pwd Print the current working directory. See **pwd**(C) for usage and description.

read [*name* ...]
One line is read from the standard input and, using the internal field separator, IFS (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using \new-line. Characters other than new-line can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an 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, a list of all readonly names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**-aefhkntuvx** [*arg* ...]]
The following options can be used with the **sh** command directly, as well as with the **set** command:

-a Marks variables that are modified or created for export.

-e If the shell is noninteractive, exits immediately if a command exits with a nonzero exit status.

-k Places all keyword arguments in the environment for a command, not just those that precede the command name.

-f Disables file name generation.

-h Locates and remembers function commands as functions are defined (function commands are normally located when the function is executed).

-n Reads commands but does not execute them.

-t Exits after reading and executing one command. Intended for use by C programs only; not useful interactively.

-u Treats unset variables as errors when substituting.

-v Prints shell input lines as they are read.

-x Prints commands and their arguments as they are executed.

-- Does not change any of the flags; useful in setting $1 to -.

Using + rather than - causes these flags to be turned off. These flags can also be used when invoking the shell. The current set of flags, including those listed under "Invocation," which follows, may be found in $-. The remaining arguments are positional parameters and are assigned, in order, to $1, $2, ... If no arguments are given, the values of all names are printed.

shift [*n*]
The positional parameters from $*n*+1 ... are renamed $1 .... If *n* is not given, it is assumed to be 1.

test Evaluates conditional expressions. See **test**(C) for usage and description.

times Prints the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*]
The command *arg* is read and executed when the shell receives signal(s) *n*. (*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. The highest signal number allowed is 16. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11

(memory fault) produces an error, because the shell uses this signal internally. If *arg* is absent, then all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]
Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You may lower your own **ulimit**, but only the super-user (see **su**(C)) can raise a **ulimit**.

umask [*nnn*]
The user file-creation mask is set to *nnn* (see **umask**(C)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables PATH, PS1, PS2, MAILCHECK and IFS cannot be unset.

wait [*n*]
Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through **exec**(C) and the first character of argument zero is -, commands are initially read from /etc/profile and from $HOME/.profile, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as /bin/sh. The flags below are interpreted by the shell on invocation only.

-c *string* If the **-c** flag is present, commands are read from *string*.

-s If the **-s** flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for Special Commands) is written to file descriptor 2.

-i If the **-i** flag is present or if the shell input and output are attached to a terminal, the shell is interactive. In this case TERMINATE is ignored so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

-r If the **-r** flag is present, the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of **rsh** are identical to those of **sh**, except that the following are disallowed:

- Changing directory (see **cd**(C))
- Setting the value of $PATH

- Specifying path or command names containing /
- Redirecting output (> and >>).

The restrictions above are enforced after .profile is interpreted.

A restricted shell can be invoked in one of the following ways:

- **Rsh** is the file name part of the last entry in the /etc/passwd file (see **passwd**(M)).
- The environment variable SHELL exists and **rsh** is the file name part of its value.
- The shell is invoked; rsh is the file name part of argument 0.
- The shell is invoked with the **-r** option.

When a command to be executed is found to be a shell procedure, **rsh** invokes **sh** to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of .profile (see **profile**(M)) has complete control over user actions by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory).

The system administrator often sets up a directory of commands (i.e., /usr/rbin) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, **red**. Note that PATH must be set to something other than the default to prevent the user from simply invoking an unrestricted shell by typing **sh**.

Exit Status

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, execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the exit command described previously).

Files

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

See Also

cd(C), echo(C), env(C), getops(C), intro(C), login(C), newgrp(C), profile(M), pwd(C), test(C), umask(C), wait(C), and dup(S), exec(S), fork(S), pipe(S), signal(S), ulimit(S) in the *Reference (CP, S, F)*

User's Guide

Caveats

Words used for file names in input/output redirection are not interpreted for file name generation (see "File Name Generation," above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message "cannot fork, too many processes," try using the **wait**(C) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

Notes

If a command is executed, and a command with the same name is installed in a directory in the search path (before the directory where the original command was found), the shell will continue to **exec** the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For **wait** *n*, if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

Name

shl - Shell layer manager.

Syntax

shl

Description

Shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the **stty** option **-loblk** may be set within the layer.

The **stty**(C) character **swtch** (set to ^ if NUL) is used to switch control to **shl** from a layer. **Shl** has its own prompt, >>>, to help distinguish it from a layer. A *layer* is a shell which has been bound to a virtual tty device (**/dev/sxt???**). The virtual device can be manipulated like a real tty device using **stty**(C) and **ioctl**(S). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab, or new-line. Only the first eight characters are significant. The *names* (1) through (7) cannot be used when creating a layer. They are used by **shl** when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the **shl** prompt level. Any unique prefix is accepted.

create [*name*] Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form (#) where # is the last digit of the virtual device bound to the layer. The shell prompt variable PS1 is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block *name* [*name* ...]
For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the **stty** option **-loblk** within the layer.

delete *name* [*name* ...]
For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see **signal**(S)).

help (or **?**) Print the syntax of the **shl** commands.

layers [**-l**] [*name* ...]
For each *name*, list the layer name and its process group. The **-l** option produces a **ps**(C)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*] Make the layer references by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
: For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the **stty** option **-loblk** within the layer.

quit
: Exit **shl**. All layers are sent the SIGHUP signal.

name
: Make the layer referenced by *name* the current layer.

Files

/dev/sx???
: Virtual tty devices

$SHELL
: Variable containing path name of the shell to use (default is **/bin/sh**).

See Also

sh(C), stty(C), ioctl(S), signal(S)

Name

size - Prints section sizes in bytes of common object files.

Syntax

size [**-n**] [**-f**] [**-o**] [**-x**] [**-d**] [**-V**] *files*

Description

The **size** command produces section size information in bytes for each loaded section in the common object files (COFF). The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is input to the **size** command the information for all archive members is displayed.

The **-n** option includes NOLOAD sections in the size.

The **-f** option produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.

The **-d** option prints numbers in decimal (the default). The **-o** or **-x** option prints in octal or in hexadecimal, respectively.

The -V option will supply the version information on the **size** command.

See Also

as(CP), cc(CP), ld(CP), ar(F) in the *Reference (CP, S, F)*

Notes

Since the size of bss sections is not known until link-edit time, the **size** command will not give the true total size of pre-linked objects.

Diagnostics

size: *name:* cannot open
if *name* cannot be read.

size: *name:* bad magic
if *name* is not an appropriate common object file.

Name

sizefs - Determines the size of a logical device from the layout information associated with a hard disk.

Syntax

sizefs *layout-file logical-device-number*

Description

The **sizefs** command prints the size in sectors of the area on the disk you specify. It gets its information out of the structure created by the **layout**(C) command. Its most common use is in shell scripts to create a file system on the hard disk, where the size of the root partition is used as an argument to **mkfs**(M) or **archive**(C).

Logical device number 0 is the size of the entire disk (excluding bad blocks). For example,

sizefs /dev/hd0.layout 2

returns the size of the root file system.

Related Commands

layout(C), mkfs(M)

Name

sleep - Suspends execution of a command for a specified interval.

Syntax

sleep *time*

Description

The **sleep** command suspends execution of a command for a specified number of 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
```

Time must be less than 4,294,967,295 (2^{32}-1) seconds for the Series 386.

Name

sort - Sorts and merges files.

Syntax

sort [**-cmu**] [**-o***output*] [**-T***directory*] [**-y***kmem*] [**-z***recsz*] [**-dfiMnr**] [**-btx**] [*+pos*] [*-pos*] [*file ...*]

Description

The **sort** command merges and sorts lines from all named *files* and writes the result on the standard output. A dash (-) may appear as a file in the *files* argument signifying 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 options you specify with the command.

Options

The following options alter output behavior:

-c Checks that the input file is sorted according to the ordering rules; gives no output unless the file is out of sort.

-m Merges only, the input files are already sorted.

-u Suppresses all but one in each set of duplicated lines. Ignored bytes and bytes outside keys do not participate in this comparison.

-o*output* Uses *output* file instead of the standard output. This file may be the same as one of the inputs. There may be ptional blanks between **-o** and *output*.

-T*directory*
Uses *directory* as a temporary directory (instead of /usr/tmp or /tmp) when doing the sort.

-y*kmem* The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, **sort** begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, **sort** will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding limit will be used. By convention, -y (with no argument) starts with maximum memory.

-z*recsz* The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **-c** or **-m** options, a popular system default size will be used. Lines longer than the buffer size will cause **sort** to terminate abnormally. Supplying *recsz* the actual number of bytes in the longest line to be merged (or some larger value) will prevent termination.

The following options override the default ordering rules:

-d "Dictionary" order: only letters, digits and blanks are significant in comparisons.

-f Folds uppercase letters onto lowercase.

-i Ignores characters outside the ASCII octal range 040 - 0176 in non-numeric comparisons.

-M Compare as months. The first three non-blank characters of the field are folded to uppercase and compared so that "JAN" < "FEB" < ...< "DEC." Invalid fields compare low to "JAN." This option implies **-b.**

-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 Reverses the sense of comparisons.

The notation *+pos1 -pos2* restricts a sort key to a field beginning at *pos1* and ending at *pos2*.

NOTE

Column 0 is the starting position (*pos1* = column 0).

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the first field. The treatment of field separators can be altered using the options:

-t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field.

-b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *-pos2* argument (see below).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr.** A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect, *n* is counted from the first non-blank in the *m*+1st field; *+m.*0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *-m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means .0, indicating the last character in the *m*th field. If the **b** flag is in effect, *n* is counted from the last leading blank in the *m*+1st field; *-m.***b** refers to the first non-blank in the *m*+1st field.

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. Very long lines are silently truncated.

Examples

This command prints an alpabetized list of all the unique spellings in a list of words (capitalized words differ from uncapitalized).

sort -u +0f +0 list

This command prints the password file sorted by user ID.

sort -t: +2n /etc/passwd

This command prints the first instance of each month in an already-sorted file of month-day entries.

sort -um +0 -1 dates

Name

spell, hashmake, spellin, hashcheck - Finds spelling errors.

Syntax

spell [**-v**] [**-b**] [**-x**] [**-l**] [*+local_file*] [*files*]
/usr/lib/spell/hashmake
/usr/lib/spell/spellin *n*
/usr/lib/spell/hashcheck *spelling_list*

Description

Spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/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**(1), **tbl**(1), and **eqn**(1) constructions.

Under the **-v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **-b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *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.

By default, **spell** (like **deroff**(1)) follows chains of included files (**.so** and **.nx** **troff**(1) requests), *unless* the names of such included files begin with /usr/lib. Under the **-l** option, **spell** will follow the chains of *all* included files.

Under the *+local_file* option, words found in *local_file* are removed from **spell**'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to **spell**'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with 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 (see *FILES*). 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.

Three routines help maintain and check the hash lists used by **spell:**

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

Files

D_SPELL=/usr/lib/spell/hlist[ab]	Hashed spelling lists, American/British
S_SPELL=/usr/lib/spell/hstop	Hashed stop list
H_SPELL=/usr/lib/spell/spellhist	History file
/usr/lib/spell/spellprog	Spell program

See Also

sed(C), sort(C), tee(C), and dtroff(1)
eqn(1), tbl(1), troff(1) in the *DOCUMENTER'S WORKBENCH*

Bugs

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

Name

spline - Interpolates smooth curve.

Syntax

spline [*options*]

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 has two continuous derivatives, and enough points to look smooth when plotted.

The following *options* are recognized, each as a separate argument:

-a Supplies 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 *num* The constant *num* used in the boundary value computation:

$$y''_0 = num\ y'_1, \ldots, y''_n = num\ y'_{n-1}$$

is set by the next argument. By default *num* = 0.

-n *num* Spaces output points so that approximately *num* intervals occur between the lower and upper *x* limits. *Num* is a positive interger; the default is 100.

-p Makes 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).

Diagnostics

When data is not strictly monotone in *x*, **spline** reproduces the input without interpolating extra points.

Notes

A limit of 1000 input points is silently enforced.

Name

split - Splits a file into pieces.

Syntax

split [*-n*] [*file* [*name*]]

Description

Split reads *file* and writes it in *n*-line pieces (default 1000) 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 used by default.

If no input file is given, or if a dash (-) is given instead, the standard input file is used.

See Also

bfs(C), csplit(C)

Name

ssp - Removes consecutive blank lines.

Syntax

ssp [-] [*file ...*]

Description

Ssp compresses consecutive blank lines to at most one blank line to produce more compact output for a CRT terminal or save paper on the printer. If the optional - is given, ssp is even more zealous, and gets rid of all empty lines.

Name

strings - Finds the printable strings in an object file.

Syntax

strings [-] [-o] [*-number*] *file ...*

Description

Strings looks for ASCII strings in a binary file. A string is any sequence of four or more printing characters ending with a newline or a null character. Unless the - flag is given, **strings** only looks in the initialized data space of object files.

-o Each string is preceded by its decimal offset in the file.

-number Uses *number* as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

See Also

hd(C), od(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Name

stty - Sets the options for a terminal.

Syntax

stty [**-a**] [**-g**] [*options*]

Description

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

See **xtty**(C) for **stty** extensions.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding control character (e.g., ^h is Ctrl-h; in this case, recall that Ctrl-h is the same as the Backspace key.) The sequence ^' means that an option has a null value.

-a Reports all of the option settings.

-g Reports current settings in a form that can be used as an argument to another **stty** command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (-parenb)	Enable (disable) parity generation and detection.
parodd (-parodd)	Select odd (even) parity.
cs5 cs6 cs7 cs8	Select character size (see **termio**(M)).
0	Hang up phone line immediately.

110 300 600 1200 1800 2400 4800 9600 19200 38400
Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)

hupcl (-hupcl) Hang up (do not hang up) dataphone connection on last close.

hup (-hup) Same as **hupcl (-hupcl)**.

cstopb (-cstopb) Use two (one) stop bits per character.

cread (-cread) Enable (disable) the receiver.

clocal (-clocal) Assume a line without (with) modem control.

loblk (-loblk) Block (do not block) output from a non-current layer.

Input Modes

ignbrk (-ignbrk) Ignore (do not ignore) break on input.

brkint (-brkint) Signal (do not signal) INTR on break.

ignpar (-ignpar) Ignore (do not ignore) parity errors.

parmrk (-parmrk) Mark (do not mark) parity errors (see **termio**(M)).

inpck (-inpck) Enable (disable) input parity checking.

istrip (-istrip) Strip (do not strip) input characters to seven bits.

inlcr (-inlcr) Map (do not map) NL to CR on input.

igncr (-igncr) Ignore (do not ignore) CR on input.

icrnl (-icrnl) Map (do not map) CR to NL on input.

iuclc (-iuclc) Map (do not map) upper-case alphabetics to lower case on input.

ixon (-ixon) Enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (-ixany) Allow any character (only DC1) to restart output.

ixoff (-ixoff) Request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (-opost) Post-process output (do not post-process output; ignore all other output modes).

olcuc (-olcuc) Map (do not map) lower-case alphabetics to upper case on output.

onlcr (-onlcr) Map (do not map) NL to CR-NL on output.

ocrnl (-ocrnl) Map (do not map) CR to NL on output.

onocr (-onocr) Do not (do) output CRs at column zero.

onlret (-onlret) On the terminal NL performs (does not perform) the CR function.

ofill (-ofill) Use fill characters (use timing) for delays.

ofdel (-ofdel) Fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 Select style of delay for carriage returns (see **termio**(M)).

nl0 nl1 Select style of delay for line-feeds (see **termio**(M)).

tab0 tab1 tab2 tab3
Select style of delay for horizontal tabs (see **termio**(M)).

bs0 bs1 Select style of delay for backspaces (see **termio**(M)).

ff0 ff1 Select style of delay for form-feeds (see **termio**(M)).

vt0 vt1 Select style of delay for vertical tabs (see **termio**(M)).

Local Modes

isig (-isig) Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.

icanon (-icanon) Enable (disable) canonical input (ERASE and KILL processing).

xcase (-xcase) Canonical (unprocessed) upper/lower-case presentation.

echo (-echo) Echo back (do not echo back) every character typed.

echoe (-echoe) Echo (do not echo) ERASE character as a backspacespace-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (-echok) Echo (do not echo) NL after KILL character.

lfkc (-lfkc) The same as **echok (-echok)**; obsolete.

echonl (-echonl) Echo (do not echo) NL.

noflsh (-noflsh) Disable (enable) flush after INTR, QUIT, or SWTCH.

Control Assignments

control-character c
: Set *control-character* to *c*, where *control-character* is **erase, kill, intr, quit, swtch, eof, ctab, min,** or **time** (**ctab** is used with **-stappl**; **min** and **time** are used with **-icanon**; see **termio**(M)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding Ctrl character (e.g., ^d is a Ctrl-d); ^? is interpreted as Del and ^- is interpreted as undefined.

line *i*
: Set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity**
: Enable **parenb** and **cs7.**

oddp
: Enable **parenb, cs7,** and **parodd.**

-parity, -evenp, or **-oddp**
: Disable **parenb,** and set **cs8.**

raw (**-raw** or **cooked**)
: Enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

nl-(**-nl**)
: Unset (set) **icrnl, onlcr.** In addition **-nl** unsets **inlcr, igncr, ocrnl,** and **onlret.**

lcase (**-lcase**)
: Set (unset) **xcase, iuclc,** and **olcuc.**

LCASE (**-LCASE**)
: Same as **lcase** (**-lcase**).

tabs (**-tabs** or **tab3**)
: Preserve (expand to spaces) tabs when printing.

ek
: Reset ERASE and KILL characters back to normal **#** and **@.**

sane	Resets all modes to some reasonable values.
term	Set all modes suitable for the terminal type *term*, where *term* is one of **tty33, tty37, vt05, tn300, ti700,** or **tek.**

See Also

tabs(C), termio(M) and ioctl(S) in the *Reference (CP, S, F)*

Name

su - Become super-user or another user without logging off.

Syntax

su [-] [*name* [*arg* ...]]

Description

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user). If a user name has a password, **su** prompts for the user's password.

To use **su**, the appropriate password must be supplied (unless one is already **root**). If the password is correct, **su** will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see **passwd**(M)), or /bin/sh if none is specified (see **sh**(C)). To restore normal user ID privileges, exit the new shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like **sh**(C). If the first argument to **su** is a -, the environment will be changed to what would be expected if you actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is -, thus causing first the system's profile (etc/profile) and then the specified user's profile (.profile in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of $PATH, which is set to /bin:/etc:/usr/bin for **root**. If the optional program used as the shell is /bin/sh, the user's .profile can check *arg0* for **-sh** or **-su** to determine if it was invoked by **login**(C) or **su**(C), respectively. If the user's program is other than /bin/sh then .profile is invoked with an *arg0* of *-program* by both **login**(C) and **su**(C).

Examples

To become user **bin** while retaining your previously exported environment, type:

su bin

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, type:

su - bin

To execute a command with the temporary environment and permissions of user **bin**, type:

su - bin -c *command args*

Files

/etc/passwd	System's password file
/etc/profile	System's profile
$HOME/.profile	User's profile
/usr/adm/sulog	Log file

See Also

env(C), login(C), sh(C), passwd(C), profile(M), environ(M)

Name

sum - Calculates checksum and counts blocks in a file.

Syntax

sum [**-r**] *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 a transmission line.

-r Causes an alternate algorithm to be used in computing the checksum.

See Also

wc(C)

Diagnostics

"Read error" is not distinguishable from end-of-file on most devices; therefore, check the block count.

Name

swap - Changes swap device configuration.

Syntax

/etc/swap -a *swapdev swaplow swaplen*
/etc/swap -d *swapdev swaplow*
/etc/swap -l

Description

Swap provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized.

-a Add the specified swap area. *Swapdev* is the name of the block special device, e.g., /dev/hd0a. *Swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. *Swaplen* is the length of the swap area in 512-byte blocks. This option can only be used by the super-user. Swap areas are normally added by the system start-up routine /etc/rc when going into multi-user mode.

-d Delete the specified swap area. *Swapdev* is the name of block special device, e.g., /dev/hd0a. *Swaplow* is the offset in 512-byte blocks into the device where the swap area begins. Using this option marks the swap area as "INDEL" (in process of being deleted). The system will not allocate any new blocks from the area, and will try to free swap blocks from it. The area will remain in use until all blocks from it are freed. This option can only be used by the super-user.

-l List the status of all the swap area. The output has four columns:

DEV The *swapdev* special file for the swap area if one can be found in the or /dev directory, and its major/minor device number in decimal. LOW The *swaplow* value for the area in 512-byte blocks.

LEN — The *swaplen* value for the area in 512-byte blocks.

FREE — The number of free 512-byte blocks in the area. If the swap area is being deleted, this column will be marked INDEL.

Warnings

No check is done to see if a swap area being added overlaps with an existing swap area or file system.

Name

sync - Updates the super block.

Syntax

sync

Description

The **sync** command executes the **sync**(S) system primitive. If the system is to be stopped, **sync** must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. **Shutdown**(M) automatically calls **sync** before shutting down the system.

Sync will only write local buffers to local disks. So, if you do a write to a file on a remote machine in an RFS environment, **sync** will not force buffers to be written out to disk on the remote machine.

See Also

sync(S) in the *Reference (CP, S, F)*

Name

sysconf - Prints system configuration information.

Syntax

sysconf [-pnrafcv]

Description

Sysconf prints system configuration information.

If you type **sysconf** without any options, you will see all of the information below.

The options are:

-p Displays the number of processors installed on the system.

-n Displays the maximum number of processes allowed to exist systemwide.

-r Displays the amount of real memory.

-a Displays the amount of available memory.

-f Displays "fp" if a floating point co-processor is installed and "fpem" if not.

-c Displays the types of communication boards installed. The logical number of the board is displayed first, followed by a ":", followed by the board type. An "m" represents a Multidrop board, an "s" represents a SIO board, and an "a" represents an ACPA board. For the 386 Series 1000, an SIO will be reported as a Multidrop. Multiple SIO boards appear only as one board (e.g., 0:m).

-v Displays the version string of the kernel.

See Also

sysconf(S)

Name

tabs - Sets tabs on a terminal.

Syntax

tabs [*tabspec*] [-T*type*] [+m*n*]

Description

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec — Four types of tab specification are accepted for *tabspec*. They are described below: canned (*-code*), repetitive (*-n*), arbitrary (*n1,n2,...*), and file (*--file*). If no *tabspec* is given, the default value is -8, i.e., operating system "standard" tabs. The lowest column number is 1. Note that for **tabs**, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code — Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

-a 1,10,16,36,72
Assembler, IBM S/370, first format

-a2 1,10,16,40,72
Assembler, IBM S/370, second format

-c 1,8,12,16,20,55
COBOL, normal format

-c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see **fspec**(F)):

<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50, 54, 58, 62, 67
COBOL compact format (columns 1-6 omitted), with more **tabs** than **-c2**. This is the recommended format for COBOL. The appropriate format specification is (see **fspec**(F)):

<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49, 53 , 57, 61
PL/1

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

-n A *repetitive* specification requests **tabs** at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value **8**: this represents the operating system "standard" tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0**, implying no tabs at all.

n1,n2,... The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1,10,20,30,** and **1,10,+10,+10** are considered identical.

--file If the name of a *file* is given, **tabs** reads the first line of the file, searching for a format specification (see **fspec**(F)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8.** This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the **pr**(C) command as follows:

tabs *--file*; **pr** *file*

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

-T*type* **Tabs** usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in **term**(M). If no **-T** flag is supplied, **tabs** uses the value of the environment variable TERM. If TERM is not defined in the *environment* (see **environ**(M)), **tabs** tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all **tabs** to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is **10.** For a TermiNet, the first value in the tab list should be **1**, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is ob-

tained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

Examples

tabs -a Example using *-code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs -8 Example of using *-n* (*repetitive* specification), where *n* is **8**, causes **tabs** to be set every eighth position: 1+(1*8), 1+(2*8),... which evaluate to columns 9, 17,...

tabs 1,8,36 Example of using *n1,n2,...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs --$HOME/fspec.list/att4425
Example of using *-file* (*file* specification) to indicate that tabs should be set according to the first line of $HOME/fspec.list/att4425 (see **fspec**(F)).

Diagnostics

illegal tabs Arbitrary tabs were ordered incorrectly.

illegal increment A zero or missing increment is found in an arbitrary specification.

unknown tab code A *canned* code cannot be found.

can't open The *--file* option was used, and file can't be opened.

file indirection The *--file* option was used and the specification in that file points to yet another file. Indirection of this form is not permitted.

See Also

pr(C), environ(M), term(M), terminfo(M), and fspec(F) in the *Reference (CP, S, F)*

Notes

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

Name

tail - Delivers the last part of a file.

Syntax

tail [±[*number*][**lbc**] [**-f**]] [*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 (if *number* is null, the value 10 is assumed). *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.

With the **-f** (follow) option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but it will enter an endless loop. In this loop, the program sleeps for a second and then attempts to read and copy further records from the input file. Thus, it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

tail -f terry

will print the last ten lines of the file, terry, followed by any lines that are appended to file between the time **tail** is initiated and killed. As another example, the command:

tail -15cf terry

will print the last 15 characters of the file terry, followed by any lines that are appended between the time **tail** is initiated and killed.

See Also

dd(C)

Notes

Tails relative to the end of the file are kept in a buffer, and thus are limited in length. Unpredictable results can occur if character special files are "tailed." The **tail** command will only tail at most the last 4096 bytes of a file regardless of its line count unless the **b** option is used.

Name

tapeutil - Utility program for a streaming tape drive.

Syntax

tapeutil [-e] [-r] ...

Description

If you invoke **tapeutil** without any arguments, the tape operations are selected using an interactive menu. If you specify command line options, the corresponding tape operation will be performed non-interactively once for each occurrence of the option. Unknown options are ignored.

Options

-e Erases the tape once.

-r Retensions the tape once.

It is recommended that you erase the tape before each **archive**(C) operation. If the tape does not stream when using **archive**, retensioning and erasing the tape will usually remedy this problem.

Related Commands

archive(C)

Examples

To retension the tape twice and erase it once, type:

tapeutil -r -r -e Retn

Name

tar - Copies files to and from the hard disk to tape or floppy disk.

Syntax

tar [crtux] [bBefFilhklmnopsvVw 0,...,7] [*arguments*] *file ...*

Description

The **tar** command saves and restores files on magnetic tape or floppy disk. **Tar's** actions are controlled by a key argument, which contains at least one function letter followed by 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, a directory name refers to the files and (recursively) subdirectories of that directory.

Tar permits a file to extend across media boundaries.

Specify the function portion of the key by one of the following letters:

c Creates a new tape; writing begins at the beginning of the tape instead of after the last file. When you use this command, all previous data is erased.

r Writes the named files at the end of the tape (only for seekable devices).

t Lists the named file each time it occurs on the tape. If no file argument is given, all of the names on the tape are listed.

u Adds the named file to the tape if it is not already there or if it has been modified since last put on the tape. This option can be slow (only for seekable devices).

x Extracts the named file from the tape. If the named file matches a directory whose contents have been written on the tape, this directory is (recursively) extracted. The owner and mode are restored (if possible). If no file argument is given, the entire content of the tape or floppy is extracted. If multiple entries specifying the same file are on the tape, the last version will overwrite all preceding versions.

In addition to the key argument function, you can use the following modifiers. Arguments to the modifiers are given in the same order as the modifiers themselves.

b Causes **tar** to use the next argument as the blocking factor for tape records. The default is 18 (a maximum of 1024 for the 386 series). Use the same blocking factor on the **x** (extract) as used on the **c** (create) option.

This option should only be used with raw magnetic tape archives (see **f** below).

Don't use the **b** option with archives that are going to be updated. 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.

B Archives all files modified after the modification date and time of the file you specify (instead of /etc/bkupdate).

Can only be used with the **I** option. Also **tar** sets the modification time of the given file after the backup is complete. The **B** option sets the modification time in the user-specified file. For example:

tar cvfbBI /dev/rct 1024 /etc/timefile ./*

The user-specified file is set to zero length when its modification date is set.

e Prevents files from being split across volumes (tapes or disks). If there is not enough room on the present volume for a given file, **tar** prompts for a new volume. This is only valid when you also specify the **k** option.

f Causes **tar** to use the next argument as the name of the archive instead of /dev/tar. If the name of the file is '-', **tar** writes to standard output or reads from standard input, whichever is appropriate. Thus, you can use **tar** to move hierarchies with the command

cd fromdir; tar cf - . | (cd todir; tar xf -)

You must use this option with magnetic tape and add-on hard disks. The default is to floppy disk.

F Causes **tar** to use the next argument as the name of a file from which succeeding arguments are taken. A dash (-) signifies that arguments are taken from the standard input.

h Archives the contents of the symbolically-linked named files. **Tar cv** will only archive linkage information; **tar chv** will archive the contents.

i *date time*
Archives all files modified after *date* and *time*. The format for *date* and *time* is:

MM/DD/YY,HH:MIN:SEC

Files modified before *date* and *time* will be skipped. Any trailing portion may be omitted. DD, HH, MIN default to 0; YY defaults to the current year. For example:

tar cvif 12/22/86,04:00:00 /dev/rct *files*

I Archives all files modified after the date and time as defined by the modification time of the file /etc/bkupdate. Also, sets the modification time of /etc/bkupdate after the backup is complete. To use a different file, see the **B** option.

k Causes **tar** to use the next argument as the size of an archive volume in kilobytes. The minimum value allowed is 250. This value must be a multiple of the blocking factor (9K by default). For tape, you can specify the block size using the **b** option. Very large files are split into "extents" across volumes. When restoring from a multivolume archive, **tar** only prompts for a new volume if a split file has been partially restored.

l Tells **tar** to notify you if the link count of a dumped file doesn't match the actual number of dumped links to that file. If this option is not specified, no error messages are printed.

m Tells **tar** not to restore the modification time; the time of extraction then becomes the modification time.

n Indicates the archive device is not a magnetic tape. The **k** option implies this. Because it can seek over files it wishes to skip, **tar** can quickly list and extract the contents of an archive. Sizes are printed in kilobytes instead of tape blocks.

o Causes extracted files to take on the user/group identifier of the user running the program, rather than those on the tape.

p Indicates that files are extracted using their original permissions. It is possible that a regular user may be unable to extract files because of the permission associated with the files or directories being extracted.

s *file*
Runs the /bin/sum algorithm on the archive and writes the resulting checksum in *file*.

v Displays 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 and path.

V Verifies the named file on the tape. **Tar** will compare the tape file to the disk file and report any file change or comparison errors. If no file argument is given, the entire contents of the tape or floppy is verified. **Tar** will exit with an exit code of 9 if there are any verify errors.

w Causes **tar** to display the action to be taken and file name, then wait for user confirmation. If you type **y**, the action is performed. Any other input causes the file to be skipped.

0,...,7

Selects the drive on which the archive is mounted. This option should only be selected if you have linked the appropriate /dev/mt to the desired device.

The floppy devices supported are:

floppy device	tracks/ inch	double or single sided	sectors/ track	size (bytes)	sectors
fd096ds15	96	double	15	1.15M	2300
fd096ds9 (default)	96	double	9	720K	1440
fd048ds9	48	double	9	360K	720
fd048ds8	48	double	8	320K	640
fd048ss9	48	single	9	180K	360
fd048ss8	48	single	8	160K	320

Files

/dev/tar Default input/output device
/tmp/tar*

Examples

This command copies the directory /usr/john to floppy disk(s).

tar cv /usr/john Retn

This command copies the files on the floppy disk to the directory /usr/john. The **cd** command is used first to make sure you are in the correct directory.

cd /usr/john Retn
tar xv Retn

This command displays the contents of the floppy disk you have in the drive.

tar tv Retn

This command pipes the **tar tv** command through the **lpr** command. This causes the contents of the floppy disk to be printed out on your serial printer.

tar tv | lpr Retn

This command copies *files* from a floppy disk device named /dev/fd096ds15 (*files* are the names of files to archive, and 1152 is the capacity of the disk in kilobytes). Arguments to key letters are given in the same order as the key letters themselves, thus the fk key letters have corresponding arguments /dev/fd096ds15 and 1152. If a file is a directory, the contents of the directory are recursively archived.

tar cvfk /dev/fd096ds15 1152 *files* Retn

This command extracts all the files with the exact same pathnames used when the archive was created.

tar xvf /dev/fd096ds15 Retn

This command copies the directory /usr/john to cartridge tape(s).

tar cvfb /dev/rct 126 /usr/john Retn

Name

tee - Creates a tee in a pipe.

Syntax

tee [**-a**] [**-i**] [*file* ...]

Description

Tee transcribes standard input to the standard output and makes copies in the *files*. The options are:

-a Causes the output to be appended to the files rather than overwriting them.

-i Ignores interrupts.

Examples

The following example illustrates the creation of temporary files at each stage in a pipeline:

grep ABC *file* **| tee ABC.grep | sort | tee ABC.sort | more**

This example shows how to **tee** output to the terminal screen:

grep ABC *file* **| tee /dev/tty | sort | uniq >final.file**

Name

test - Evaluates an expression.

Syntax

test *expr*
[*expr*]

Description

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; **test** also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second Syntax line) must be separate arguments to the **test** command; normally these items are separated by spaces.

The following primitives are used to construct *expr*:

-r *file*	True if *file* exists and is readable.
-w *file*	True if *file* exists and is writable.
-x *file*	True if *file* exists and is executable.
-f *file*	True if *file* exists and is a regular file.
-d *file*	True if *file* exists and is a directory.
-c *file*	True if *file* exists and is a character special file.
-b *file*	True if *file* exists and is a block special file.
-p *file*	True if *file* exists and is a named pipe (fifo).

-u *file* — True if *file* exists and its set-user-ID bit is set.

-g *file* — True if *file* exists and its set-group-ID bit is set.

-k *file* — True if *file* exists and its sticky bit is set.

-s *file* — True if *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 non-zero.

s1 **=** *s2* — True if strings *s1* and *s2* are identical.

s1 **!=** *s2* — True if strings *s1* and *s2* are *not* identical.

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**, and **-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 (**-a** has higher precedence than **-o**).

(expr) — Parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

See Also

find(C), sh(C)

Notes

If you test a file you own (the **-r**, **-w**, or **-x** tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The **=** and **!=** operators have a higher precedence than the **-r** through **-n** operators, and **=** and **!=** always expect arguments; therefore, **=** and **!=** cannot be used with the **-r** through **-n** operators.

If more than one argument follows the **-r** through **-n** operators, only the first argument is examined; the others are ignored, unless a **-a** or a **-o** is the second argument.

Name

tic - Compiles terminfo source.

Syntax

tic [-v[*n*]] [-c] *file*

Description

Tic translates a **terminfo**(M) file from the source format into the compiled format. The results are placed in the directory /usr/lib/terminfo. The compiled format is necessary for use with the library routines described in **curses**(S).

-v*n* Output (verbose) to standard error trace information showing **tic**'s progress. The optional integer *n* is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If *n* is omitted, the default level is 1. If *n* is specified and greater than 1, the level of detail is increased.

-c Only check *file* for errors. Errors in **use=** links are not detected.

file Contains one or more **terminfo**(M) terminal descriptions in source format (see **terminfo**(M)). Each description in the file describes the capabilities of a particular terminal. When a **use=***entry-name* field is discovered in a terminal entry currently being compiled, **tic** reads in the binary from usr/lib/terminfo to complete the entry. (Entries created from *file* will be used first. If the environment variable TERMINFO is set, that directory is searched instead of /usr/lib/terminfo.)

Tic duplicates the capabilities in *entry-name* for the current entry, with the exception of those capabilites that explicitly are defined in the current entry.

If the environment variable TERMINFO is set, the compiled results are placed there instead of /usr/lib/terminfo.

Files

/usr/lib/terminfo/?/*	Compiled terminal description data base

See Also

terminfo(M), term(M) and curses(S) in the *Reference (CP, S, F)*

Notes

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

When the **-c** option is used, duplicate terminal names will not be diagnosed; however, when **-c** is not used, they will be.

To allow existing executables from the previous release of the UNIX System to continue to run with the compiled **terminfo** entries created by the new terminfo compiler, cancelled capabilities will not be marked as cancelled within the terminfo binary unless the entry name has a '+' within it. (Such terminal names are only used for inclusion within other entries via a **use=** entry. Such names would not be used for real terminal names.)

For example:

```
4415+nl, kf1@, kf2@, ...

4415+base, kf1=\EOc, kf2=\EOd, ...

4415-nl 4415 terminal without keys,
    use=4415+nl, use=4415+base,
```

The above example works as expected; the definitions for the keys do not show up in the 4415-nl entry. However, if the entry 4415+nl did not have a plus sign within its name, the cancellations would not be marked within the compiled file and the definitions for the function keys would not be cancelled within 4415-nl.

Diagnostics

Most diagnostic messages produced by **tic** during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

mkdir ... returned bad status
: The named directory could not be created.

File does not start with terminal names in column one
: The first thing seen in the file, after comments, must be the list of terminal names.

Token after a seek not NAMES
: Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
: or

Out of memory
: Not enough free memory was available (malloc(S) failed).

Can't open ...
: The named file could not be created.

Error in writing ...
: The named file could not be written to.

Can't link ... to ...
: A link failed.

Error in re-reading compiled ...
: The compiled file could not be read back in.

Premature EOF
: The current entry ended prematurely.

Backspaced off beginning of line
: This error indicates something wrong happened within **tic.**

Unknown Capability - "..."
: The named invalid capability was found within the file.

Wrong type used for capability "..."
For example, a string capability was given a numeric value.

Unknown token type
Tokens must be followed by '@' to cancel, ',' for booleans, '#' for numbers, or '=' for strings.

"...": bad term name
or
Line ...: Illegal terminal name - "..."
Terminal names must start with a letter or digit
The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

"...": terminal name too long.
An extremely long terminal name was found.

"...": terminal name too short.
A one-letter name was found.

"..." filename too long, truncating to "..."
The given name was truncated to 14 characters due to the operating system file name length limitations.

"..." defined in more than one entry. Entry being used is "...".
An entry was found more than once.

Terminal name "..." synonym for itself
A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
At least one of the names of the terminal should begin with a letter.

Illegal character - "..."
The given invalid character was found in the input file.

Newline in middle of terminal name
The trailing comma was probably left off of the list of names.

Missing comma
A comma was missing.

Missing numeric value
The number was missing after a numeric capability.

NULL string value
The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
Self-explanatory

Unknown option. Usage is:
An invalid option was entered.

Too many file names. Usage is:
Self-explanatory

"..." non-existent or permission denied
The given directory could not be written into.

"..." is not a directory
Self-explanatory

"...": Permission denied
Access denied.

"...": Not a directory
Tic wanted to use the given name as a directory, but it already exists as a file

SYSTEM ERROR!! Fork failed!!!
A **fork**(S) failed.

Error in following up use-links. Either there is a loop in the links or they reference non-existant terminals. The following is a list of the entries involved:

A **terminfo**(M) entry with a **use**=*name* capability either referenced a non-existant terminal called *name* or *name* somehow referred back to the given entry.

Name

time - Times a command.

Syntax

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 times are printed on the standard error output.

See Also

times(S)

Name

tk - Paginator for the Tektronix 4014.

Syntax

tk [*-N*] [**-p***L*] [**-t**] [*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. 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:

-N Divide the screen into *N* columns and wait after the last column.

-p*L* Set page length to *L* lines.

-t Don't wait between pages (for directing output to a file).

See Also

pr(C)

Name

touch - Updates access and modification times of a file.

Syntax

touch [**-amc**] [*mmddhhmm[yy]*] *file ...*

Description

Touch causes the access and modification times of each argument to be updated. If no time is specified (see **date**(C)) the current time is used.

-a Causes **touch** to update only the access time (default is **-am**).

-c Silently prevents **touch** from creating the file if it did not previously exist.

-m Updates only the modification time (default is **-am**).

The return code from **touch** is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

See Also

date(C), settime(C) and utime(S) in the *Reference (CP, S, F)*

Name

tput - Queries terminfo database.

Syntax

tput [-T*type*] *capname*

Description

The **tput** command uses the **terminfo**(M) database to make terminal-dependent capabilities and information available to the shell. This command outputs a string if the attribute (capability name) is of type **string,** or an integer if the attribute is of type **integer.** If the attribute is of type **boolean, tput** simply sets the exit code (0 for TRUE, 1 for FALSE), and produces no output.

-T*type* Indicates the type of terminal. Normally this flag isis unnecessary, as the default is taken from the environment variable **TERM.**

capname Indicates the attribute from the **terminfo** database. (See **terminfo**(M).)

Examples

This command echoes a clear-screen sequence for the current terminal.

tput clear Retn

This command prints the number of columns for the current terminal.

tput cols Retn

This command prints the number of columns for the 450 terminal.

tput -T450 cols Retn

This command sets the shell variable "bold" to stand-out mode sequence for the current terminal.

bold=`tput smso` Retn

This might be followed by a prompt:

```
echo "${bold}Please type in you name: \c"
```

This command sets the exit code to indicate if the current terminal is a hardcopy terminal.

tput hc Retn

Files

/usr/lib/terminfo/?/*	Terminal descriptor files
/usr/include/term.h	Definition files
/usr/include/curses.h	

Name

tr - Translates characters.

Syntax

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*. Any combination of the **-cds** options may be used:

-c Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

-d Deletes all input characters in *string1*.

-s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[*a-z*] Stands for the string of characters whose ASCII codes run from character *a* to character *z*, inclusive.

[*a***n*] Stands for *n* repetitions of *a*. If the first digit of *n* is 0, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character, \, may be used in the shell to remove special meaning from any character in a string. Also, a \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

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 strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline:

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

See Also

ascii(M), ed(C), sh(C)

Notes

Tr won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

Name

tra - Copies out a file as it grows.

Syntax

tra [-] [*-interval*] [*+limit*] *file*

Description

Tra will copy out the contents of a growing file to the standard output as it grows. It alternately copies out the new material in the file and sleeps for *interval* seconds; the default interval is 15 seconds. *Limit* can be given to limit the total running time of the **tra**; the default is effectively infinite.

Tra normally copies out all the text currently in *file* before beginning to watch for new text. The option "-" alone causes only new material to be copied out.

Tra is particularly useful for alternately watching the output file being written by a long shell script or a long-running program and doing real work.

See Also

tail(C)

Name

true - Returns with a zero exit value.

Syntax

true

Description

The **true** command returns with a zero exit value. **False**, **true**'s counterpart, returns a non-zero exit value. **True** is typically used in shell procedures such as:

```
while true
do
        command
done
```

Related Commands

sh(C), false(C)

Name

tset - Sets terminal modes.

Syntax

tset [-] [**-eEhkrsIQS**] [**-m** *[ident][test baudrate]:type*] [*type*]

Description

The **tset** command causes terminal-dependent processing such as setting erase and kill characters or setting and resetting delays. It is driven by the /etc/ttytype and /usr/lib/terminfo/* files.

Ports for which the terminal type is indeterminate are identified in /etc/ttytype as dialup, plugboard, etc. The port name is determined by a ttyname call on the diagnostic output. If the port is not found in /etc/ttytype the terminal type is set to unknown.

Options

- Prints the terminal type on the standard output; this can be used to get the terminal type by saying:

  set termtype = `tset -`

  If no other options are given, **tset** operates in "fast mode" and only outputs the terminal type, bypassing all other processing.

-e*c* Sets the erase character to be the character *c* on all terminals. To override this option, enter **-e#**. The default for *c* is the backspace character on the terminal, usually Ctrl-H.

-E*c* Operates only on terminals that can backspace (same as **-e**).

-h Forces **tset** to search /etc/ttytype for information, and to overlook the environment variable, TERM.

-I Suppresses outputting the terminal initialization strings.

-k*c* Sets the kill character to *c* (Ctrl-U is the default for *c*). No kill processing is done if -k is not specified. In all of these flags, "^X" (where X is any character) is equivalent to Ctrl-X.

-m[*ident*][*test baudrate*]:*type*
Allows you to specify how a given serial port is to be mapped to an actual terminal type. The option applies to any serial port in /etc/ttytype whose type is indeterminate (e.g., dialup, plugboard, etc.). The *type* specifies the terminal type to be used and *ident* identifies the name of the indeterminate type to be matched. If no *ident* is given, all indeterminate types are matched. The *test baudrate* defines a test to be performed on the serial port before the type is assigned. The *baudrate* must be as defined in **stty**(C). The *test* may be any combination of: >, =, <, @, and !.

If the *type* begins with a question mark, the system asks you if you really want that type. A null response means to use that type; otherwise, you can enter another type to be used instead. (The question mark must be escaped to prevent filename expansion by the shell.)

-Q Suppresses the printing of the "Erase set to" and "Kill set to" messages.

-r Prints the terminal type on the diagnostic output.

-s Outputs the "setenv" commands (for **csh**(C)), or "export" and assignment commands (for **sh**(C)) as determined by user's login shell.

-S Outputs only the strings to be placed in the environment variables.

Tset is most useful when included in the **.login** (for **csh**(C)) or **.profile** (for **sh**(C)) file executed automatically at login, with -m mapping used to specify the terminal type you most frequently dial in on.

Examples

This command sets your terminal to the parameters contained under terminal type **gt42** in your /usr/lib/terminfo/* file.

```
tset gt42
```

This command sets the terminal parameters for an adm3a terminal if a dialup line is used; otherwise, it prompts for the terminal type.

```
tset -m dialup\>300:adm3a -m unknown:\? -e^Z  -k^U
```

where

-m	mapping flag
dialup	identifier
\	tells the system to take the next character literally
>300	speed of the baud rate test
adm3a	terminal type
unknown	specifies "unknown" terminal type
?	tells the system to ask the user if he really wants that terminal type
-e^Z	sets the erase character to Ctrl-Z
-k^U	sets the kill character to Ctrl-U

To use the information created by the -s option for the Bourne shell, (sh(C)), repeat these commands:

```
tset -s...> /tmp/tset$$
/tmp/tset$$
rm -f /tmp/tset$$
```

To use the information for **csh**(C), use the following:

```
set noglob
set term=('tset -S....')
setenv TERM $term[1]
setenv TERMCAP "$term[2]"
unset term
unset noglob
```

Files

/etc/ttytype	Port name to terminal type map database
/usr/lib/terminfo	Terminal capability database

See Also

stty(C), ttys(M), termcap(M)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Name

tsort - Sorts a file topologically.

Syntax

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 assumed.

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(CP) in the *Reference (CP, S, F)*

Diagnostics

Odd data: There is an odd number of fields in the input file.

Notes

The sort algorithm is quadratic, which can be slow if you have a large input list.

Name

tty - Gets the name of the terminal.

Syntax

tty [-l] [-s]

Description

Tty prints the path name of the user's terminal. The options are:

-l Prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.

-s Inhibits printing of the terminal path name, allowing one to test just the exit code.

The exit code is:

2 if invalid options were specified,
0 if standard input is a terminal,
1 otherwise.

Diagnostics

"not on an active synchronous line" if the standard input is not a synchronous terminal and -l is specified.

"not a tty" if the standard input is not a terminal and -s is not specified.

(BLANK)

Name

ua - User administration.

Syntax

ua [-h]

Description

You must be the super-user to access these commands.

Use the **ua** command for the addition, deletion, and modification of users and groups. It provides an effective means for maintaining the system password (/etc/passwd) and system group (/etc/group) files.

The command is implemented using the **termcap**(M) and **curses**(S) facilities from UC Berkeley. It must be run interactively from a terminal defined in /etc/termcap.

Options

-h Displays the program's current version and copyright notice as well as a short description of the program's functions.

After you enter the command, **ua** displays its legal commands at the top of the screen. Select a command and enter the first letter of the command at the "Command?" prompt at the bottom of the screen. Full command words are not acceptable as input. The case of each word is significant: "group" is not the same as "Group."

The **ua** screen commands are summarized as follows:

Add Adds a new user or group. After you specify the user or group and a new name, the system immediately enters the **change** command to allow modification of the new entry. At the conclusion of the **change** command, the addition is made. If a directory already exists for a new user, it is not removed. All files under /etc/newuser are copied to the new directory during the user in-

stallation process. Typically, /etc/newuser will contain the standard versions of the following files: .cshrc, .login, .logout, .profile.

The initial value given to a new user ID is one more than the maximum user ID currently in use. The same is true for a new group ID.

Delete Deletes an existing user or group. When you delete a user, the files and directories in that user's home directory will be deleted, but any other files owned by this user in the system will not be deleted. Thus, some files may have an "unknown" owner after a user is deleted. And, if a user is later added with the same user ID as the deleted user, these files will suddenly belong to the new user. The same problem may arise with the deletion and later addition of a group.

Show Shows an individual user or group or all users or groups. The word "show" may be omitted if desired.

Change Modifies any existing user or group. When you select this command, a menu appears so you can select the item to be modified. Typing Retn o Line Feed at a field change request empties th field. When you change a user or group, the corresponding entry in /etc/passwd is also changed.

Changing a user's directory causes a renaming of that directory. Make sure that the entry in /etc/passwd remains consistent.

If you want to move a user's directory from one file system to another, use **cp -r** to copy the user's directory to a new directory.

Help Displays a short informative text on the screen, explaining each of the commands. "?" is equivalent to help. The message is the same one you get when you enter the **ua** command with the "-h option.

! Escapes to the shell (see the **sh**(C) command).

! Escapes to the shell (see the **sh**(C) command). If no arguments are given, a shell is invoked which will continue until it receives an end-of-file. Then **ua** resumes. If arguments are present, a shell is invoked with the "-c" option and the arguments are passed along. **Ua** resumes immediately thereafter.

Quit Immediately terminates **ua** and returns to the system.

Any command that is not understood by **ua** causes an appropriate error message to be displayed. As a side-effect, the working portion of the screen is cleared.

The **ua** command does not distinguish between Retn and Line Feed. They may be used interchangeably.

If the screen becomes "dirty" for some reason, you can force **ua** to clear it and redisplay the current contents by transmitting an ASCII DC2. This is Ctrl-R on most terminals.

The **ua** command understands the Backspace function (as obtained from /etc/termcap). In addition, any time a word is partially formed, the Esc key will cause the partial word to be discarded and input restarted.

The **ua** command interprets the Cancel key to mean "terminate the current operation." The Cancel key is Ctrl-X on most terminals. The Cancel key is more powerful than Esc, but not as powerful as "interrupt."

The **ua** command will immediately return to the top-level command interpreter upon receipt of an interrupt signal. Such a signal is usually generated by Del or Break.

The **ua** command creates a special user named "standard" in /etc/passwd if one is not already present. This entry is used as the template for installing new users. Thus, if you want all new users defaulted to the standard shell (/bin/sh) for the shell field, it is only necessary to update the shell field in the "standard" user.

Before adding a new user with a new group, the new group should be added. Otherwise, **ua** has no way to properly create the new entry in /etc/passwd since it contains group numbers rather than group names.

During program initialization **ua** copies /etc/passwd and /etc/group to /etc/opasswd and /etc/ogroup, respectively. Thus, if a mistake or disaster occurs during the use of this program, the user may recover the prior state of either or both files.

Files

/etc/passwd	Used for login name to user ID conversions
/etc/group	Used for group name to group ID conversion
/etc/opasswd	This file is a copy of /etc/passwd before any modifications are made
/etc/ogroup	This file is a copy of /etc/group before any modifications are made
/etc/newuser	Directory containing files which will be installed in a new user's account
/etc/termcap	Contains terminal attribute descriptions
/tmp/passwd	Temporary file
/tmp/group	Temporary file /etc/ua.lock lock file

See Also

group(M), passwd(M)
Operations Guide

Name

umask - Sets file-creation mode mask.

Syntax

umask [*nnn*]

Description

The user file-creation mode mask is set to *nnn*. The three octal digits refer to read/write/execute permissions for owner, group, and others, respectively. Only the low-order 9 bits of cmask and the file mode creation mask are used. The value of each specified digit is "subtracted" from the corresponding "digit" specified by the system for the creation of any file (see the **create** files). This is actually a binary masking operation, and thus the name "umask." In general, binary ones remove a given permission and zeros have no effect at all. For example, **umask 022** removes group and others write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *nnn* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell. By default, login shells have a umask of 022.

Umask can be included in the user's **.profile** and invoked at login to automatically set the user's permissions on files or directories created.

Related Commands

chmode(C), sh(C), and chmod(S), creat(S), umask(S), profile(F) in the *Reference (CP, S, F)*

(BLANK)

Name

uname - Displays the current operating system information.

Syntax

uname [*options*]

Description

The **uname** command displays the current operating system name on the standard output file.

Options

-a Displays all information on the screen

-d Displays OEM (Distributor) for this system

-h Displays the hardware machine name

-i Displays an integer representing the machine type:

1 = Altos 586/986
2 = Altos 186
3 = Altos 1086/2086
6 = Altos 486
7 = Altos 586T/986T
8 = Altos 686/886
9 = Altos 3086
10 = Altos 386, Series 2000
11 = Altos 386, Series 1000

-m Displays original supplier (Manufacturer) of the system

-n Displays the nodename (may be a name by which the system is known to a communications network)

-r Displays the operating system release

-s Displays the system name (default)

-S *nodename*
Sets the name to *nodename* (for remote communications). The *nodename* is also written to the /etc/systemid file. You must be the super-user to use this option. Do not use -S to set the name if Worknet is running.

-u Displays user serial number for this system

-v Displays the operating system version number

Files

/etc/systemid	Contains the name of sites as defined with the -N option (for compatibility with older programs).

Name

uniq - Reports repeated lines in a file.

Syntax

uniq [**-cdu** [*+n*] [*-n*]] [*input* [*output*]]

Description

Uniq reads the input file and compares adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see **sort**(C). The options are:

-c Generates an output report in default style with each line preceded by a count of the number of times it occurred (supersedes the **-u** and **-d** options).

-d Specifies that one copy of the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

-u Prints only the lines that are not repeated in the original file.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of characters without spaces and tabs separated by spaces and tabs from adjacent fields.

+n The first *n* characters are ignored. Fields are skipped before characters.

See Also

comm(C), sort(C)

Name

units - Converts units.

Syntax

units

Description

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have:  inch
You want:  cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of **units** preceded optionally by a numeric multiplier. Powers are indicated by positive integer suffixes; division is indicated by the standard sign:

```
You have:  15 lbs force/in2
You want:  atm
        * 1.020689e+00
        / 9.797299e-01
```

Units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, as well as the following:

pi	Ratio of circumference to diameter
c	Speed of light
e	Charge on an electron
g	Acceleration of gravity
force	Same as g

mole Avogadro's number

water Pressure head per unit height of water

au Astronomical unit

Pound is not recognized as a unit of mass; lb is. Compound names are run together, (e.g., lightyear). British units that differ from their US counterparts are prefixed with "br". For a complete list of **units**, type:

cat /usr/lib/unittab

Files

/usr/lib/unittab

Name

upgrade.hd - Upgrades an additional hard disk.

Syntax

upgrade.hd [2] [3]

Description

The **upgrade.hd** command is a shell script that upgrades an additional hard disk if your system supports more than one add-on hard disk. You must be the super user to use this command.

If you don't specify the number (i.e., 2 or 3) on the command line, the script prompts you for the number. Once you reply with a correct number, the system updates the additional hard disk.

Then **upgrade.hd** labels the hard disk drive and adds the **mount** entry command to the /etc/fstab file (so the add-on hard disk is mounted each time that the **multiuser** command is run).

The directory used for the add-on hard disk is /usr2 for the second hard disk, /usr3 for the third hard disk, and so on. The add-on hard disk remains mounted as /usr2 (/usr3) when **upgrade.hd** exits and whenever the system is in multiuser mode.

Related Commands

add.hd(M), add.hd(C), buildmap(C), sizefs(C)

Name

uucp, uulog, uuname - UNIX-to-UNIX system copy.

Syntax

uucp [*options*] *source-files destination-file*
uulog [*options*] **-s***system*
uulog [*options*] *system*
uulog [*options*] **-f***system*
uuname [**-l**] [**-c**]

Description of uucp

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!*path-name*

where *system-name* is taken from a list of system names that **uucp** knows about. The *system-name* may also be a list of names such as:

system-name!*system-name*!...!*system-name*!*path-name*

in which case an attempt is made to send the file via the specified route, to the destination. See the "Notes" section that follows for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information.

The shell metacharacters **?**, ***** and **[...]** appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

- A full path name
- A path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user's login directory

- A path name preceded by ~/*destination* where *destination* is appended to /usr/spool/uucppublic. NOTE: This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example, ~/dan/ as the destination will make the directory /usr/spool/uucppublic/dan if it does not exist and put the requested file(s) in that directory

- Anything else is prefixed by the current directory

If the result is an erroneous path name 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**(C)).

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

-c Do not copy local file to the spool directory for transfer to the remote machine (default).

-C Force the copy of local files to the spool directory for transfer.

-d Make all necessary directories for the file copy (default).

-f Do not make intermediate directories for the file copy.

-g*grade* *Grade* is a single letter/number; lower ascii sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the job identification ASCII string on the standard output. This job identification can be used by **uustat** to obtain the status or terminate a job.

-m Send mail to the requester when the copy is com pleted.

-n*user* Notify *user* on the remote system that a file was sent.

-r Do not start the file transfer, just queue the job.

-s*file* Report status of the transfer to *file*. Note that the *file* must be a full path name.

-x*debug_level*
Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

Description of uulog

Uulog queries a log file of **uucp** or **uuxqt** transactions in a file:

/usr/spool/uucp/.Log/uucico/*system*

or

/usr/spool/uucp/.Log/uuxqt/*system*

The options cause **uulog** to print logging information:

-s*sys* Print information about file transfer work involving system *sys*.

-f*system* Does a **tail -f** of the file transfer log for *system*. (You must press Break/Del to exit this function.) Other options used in conjunction with the above:

-x Look in the **uuxqt** log file for the given system.

-*number* Indicates that a **tail** command of *number* lines should be executed.

Description of uuname

Uuname lists the names of systems known to **uucp.** The -c option returns the names of systems known to **cu.** (The two lists are the same, unless your machine is using different *systems* files for **cu** and **uucp.** See the Sysfiles file.) The -l option returns the local system name.

Files

/usr/spool/uucp	Spool directories
/usr/spool/uucppublic/*	Public directory for receiving and sending (/usr/spool/uucppublic)
/usr/lib/uucp/*	Other data and program files

See Also

mail(C), uustat(C), uux(C), uuxqt(M), chmod(C), uuto(C), uupick(C)

Notes

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 path name; 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 path names. As distributed, the remotely accessible files are those whose names begin /usr/spool/uucppublic (equivalent to ~/).

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.

The forwarding of files through other systems may not be compatible with the previous version of **uucp.** If forwarding is used, all systems in the route must have the same version of **uucp.**

Protected files and files that are in protected directories that are owned by the requestor can be sent by **uucp**. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

If you use the C-shell to execute **uucp** commands, be sure to escape any exclamation point (!) used on the command line.

Name

uustat - Uucp status inquiry and job control.

Syntax

uustat [-a]
uustat [-m]
uustat [-p]
uustat [-q]
uustat [-k*jobid*]
uustat [-r*jobid*]
uustat [-ss*ystem*] [-u*user*]

Description

Uustat will display the status of, or cancel, previously specified **uucp** commands, or provide general status on **uucp** connections to other systems. Only one of the following options can be specified with **uustat** per command execution:

-a Output all jobs in queue.

-m Report the status of accessibility of all machines.

-p Execute a **ps -flp** for all the process-ids that are in the lock files.

-q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of C or X files, it is the age in days of the oldest C./X. file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts.

NOTE

For systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute.

An example of the output produced by the **-q** option is:

```
eagle    3C  04/07-11:07  NO DEVICES AVAILABLE
mh3bs3   2C  07/07-10:42  SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

-k*jobid* Kill the **uucp** request whose job identification is *jobid.* The killed **uucp** request must belong to the person issuing the **uustat** command unless one is the super-user.

-r*jobid* Rejuvenate *jobid.* The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the daemon.

Either or both of the following options can be specified with **uustat:**

-s*sys* Report the status of all **uucp** requests for remote system *sys.*

-u*user* Report the status of all **uucp** requests issued by *user.*

Output for both the **-s** and **-u** options has the following format:

```
eaglen0000  4/07-11:01:03  (POLL)
eagleN1bd7  4/07-11:07     Seagledan522/usr/dan/A
eagleC1bd8  4/07-11:07     Seagledan59 D.3b2a12ce4924
            4/07-11:07     Seagledanrmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the

user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (**rmail** - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2a1ce4924) that is created for data files associated with remote executions (**rmail** in this example).

When no options are given, **uustat** outputs the status of all **uucp** requests issued by the current user.

Files

/usr/spool/uucp/* Spool directories

See Also

uucp(C)

Name

uuto, uupick - Public UNIX-to-UNIX system file copy.

Syntax

uuto [*options*] *source-files destination*
uupick [**-s** *system*]

Description

Uuto sends *source-files* to *destination*. **Uuto** uses the **uucp**(C) facility to send files, while it allows the local system to control the file access. A *source-file* name is a path name on your machine. *Destination* has the form:

system!user

where *system* is taken from a list of system names that **uucp** knows about (see **uuname**(C)). *User* is the login name of someone on the specified system.

Two *options* are available:

-p Copy the source file into the spool directory before transmission.

-m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the **uucp** source. By default this directory is /usr/spool/uucppublic. Specifically the files are sent to:

PUBDIR/**receive**/*user*/*mysystem*/*files*

The destined recipient is notified by **mail**(C) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, **uupick** searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from system: [file file-name] [dir dirname] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

Retn	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir*. If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (Ctrl-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick when invoked with the **-s***system* option will only search the PUBDIR for files sent from *system*.

Files

PUBDIR /usr/spool/uucppublic	Public directory

See Also

mail(C), uucp(C), uustat(C), uux(C), uucleanup(M)

Notes

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

If you use the C-shell to execute **uuto,** be sure to escape any exclamation point (!) used on the command line.

Name

uux - Executes a command on a remote UNIX system.

Syntax

uux [*options*] *command-string*

Description

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

NOTE

For security reasons, most installations limit the list of commands executable on behalf of an incoming request from **uux**, permitting only the receipt of mail (see **mail**(C)). (Remote execution permissions are defined in /usr/lib/uucp/Permissions.)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of:

- A full path name
- A path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory
- Anything else is prefixed by the current directory

As an example, the command:

uux "!diff usg!/usr/dan/file1
pwba!/a4/dan/file2>!~/dan/file.diff"

will get the *file1* and *file2* files from the "usg" and "pwba" machines, execute a **diff**(C) command and put the results in file.diff in the local PUBDIR/dan/ 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.

Uux will attempt to get all files to the execution system.

For files that are output files, the file name must be escaped using parentheses. For example, the command:

uux a!cut -f1 b!/usr/file \(c!/usr/file\)

gets /usr/file from system "b" and sends it to system "a," performs a **cut** command on that file, and sends the result of the **cut** command to system "c."

Uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **-n** option. The response comes by remote mail from the remote machine.

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

- The standard input to **uux** is made the standard input to the *command-string*.

-a*name* Use *name* as the user identification replacing the initiator user ID. (Notification will be returned to the user.)

-b Return whatever standard input was provided to the **uux** command if the exit status is non-zero.

-c Do not copy local file to the spool directory for transfer to the remote machine (default).

-C Force the copy of local files to the spool directory for transfer.

-g*grade* *Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by **uustat** to obtain the status or terminate a job.

-n Do not notify the user if the command fails.

-p Same as -: The standard input to **uux** is made the standard input to the *command-string*.

-r Do not start the file transfer, just queue the job.

-s*file* Report status of the transfer in *file*.

-x*debug_level*
Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

-z Send success notification to the user.

Files

/usr/lib/uucp/spool	Spool directories
/usr/lib/uucp/Permissions	Remote execution permissions
/usr/lib/uucp/*	Other data and programs

See Also

mail(C), uucp(C), uustat(C)

Notes

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.

The execution of commands on remote systems takes place in an execution directory known to the **uucp** system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the **uux** request. The following command will NOT work:

uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz>!xyz.diff"

but the command:

uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"

will work. (If **diff**(C) is a permitted command.)

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands **u** using **uux**. However, if the requestor is root, and the directory is not searchable by "other," the request will fail.

If you use the C-shell to execute **uux**, be sure to escape any exclamation point (!) used on the command line.

Name

vi - Screen-oriented (visual) display editor based on ex.

Syntax

vi [-t *tag*] [-r *file*] [-w*n*] [-R] [+*command*] *name...*
view [-t *tag*] [-r *file*] [-w*n*] [-R] [+*command*] *name*
vedit [-t *tag*] [-r *file*] [-w*n*] [-R] [+*command*] *name*

Description

Vi (visual) is a display-oriented text editor based on an underlying line editor **ex**(C). It is possible to use the command mode of **ex** from within **vi** and vice-versa.

When using **vi**, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file. **Vi** is explained in detail in the *User's Guide*.

Invocation

The following invocation options are interpreted by **vi**:

-t *tag*	Edit the file containing the *tag* and position the editor at its definition.
-r *file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
-w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
-R	Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.

+command The specified **ex** command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The **view** invocation is the same as **vi** except that the **readonly** flag is set.

The **vedit** invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

Vi Modes

Command — Normal and initial mode. Other modes return to command mode upon completion. Esc (escape) is used to cancel a partial command.

Input — Entered by the following options: **a, i, A, I, o, O, c, C, s, S,** and **R.** Arbitrary text may then be entered. Input mode is normally terminated with Esc, or abnormally with Del (interrupt).

Last line — Reading input for : / ? or !; terminate with Retn to execute, interrupt to cancel.

Special Keys

There are several special keys in **vi.** These keys are used to edit, delimit, or abort commands and command lines.

Esc — Returns to **vi** command mode, cancels partially formed commands.

Retn — Terminates **ex** commands when in **ex** escape mode. Also used to start a new line when in insert mode.

Interrupt — Generates an interrupt, telling the editor to stop what it is doing. Aborts any command that is executing. Often the same as the Del or **Rubout** key on many terminals.

/ Specifies a string to be searched for. The slash appears on the status line as a prompt for a search string. The question mark (?) works exactly like the slash key, except that it searches backward in a file instead of forward.

: Prompts for an **ex** command. You can then type any **ex** command, followed by an Esc or Retn, and the given **ex** command is executed.

The following characters are special in Insert Mode:

Backspace Backs up the cursor one character on the current line. The last character typed before the Backspace is removed from the input buffer, but remains displayed on the screen.

Ctrl-u Moves the cursor back to the first character of the insertion, and restarts insertion. (This is actually the "kill" key; so it may be different on your system.)

Ctrl-v Removes the special significance of the next typed character. Use Ctrl-v to insert control characters. Line feed and Ctrl-j cannot be inserted in the text except as newline characters. Ctrl-q and Ctrl-s are trapped by the operating system before they are interpreted by **vi**, so they too, cannot be inserted as text.

Ctrl-w Moves the cursor back to the first character of the last inserted word.

Ctrl-t Inserts *shiftwidth* whitespace at the beginning of the current line with the **autoindent** option set.

Ctrl-@ If typed as the first character of an insertion, it is replaced with the last text inserted, and the insertion terminates. Only 128 characters are saved from the last insertion. If more than 128 characters were inserted, then this command inserts no characters. A **Ctrl-@** cannot be part of a file, even if quoted (preceded with a backslash (\)).

Command Summary

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text* Esc	insert *text*
cw*new* Esc	change word to *new*
eas Esc	add an *s* to a word
x	delete a character
dw	delete a word
dd	delete a line
3dd	delete 3 lines
u	undo previous change
ZZ	exit **vi**, saving changes
:q! Retn	quit, discarding changes
/*text* Retn	search for *text*
^U ^D	scroll up or down
:*cmd* Retn	any **ex** or **ed** command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	z G |
scroll amount	^D ^U
repeat effect	most of the rest

Interrupting, canceling

Esc	end insert or incomplete cmd
Del	(delete or rubout) interrupts
^L	reprint screen if DEL scrambles it
^R	reprint screen if ^L is Right-arrow key

File manipulation

:w Retn	write back changes
:q Retn	quit
:q! Retn	quit, discard changes
:e *name* Retn	edit file *name*
:e! Retn	reedit, discard changes
:e+ *name* Retn	edit, starting at end
:e +*n* Retn	edit starting at line *n*
:e # Retn	edit alternate file
Ctrl-^	synonym for :e #
:w *name* Retn	write file *name*
:w! *name* Retn	overwrite file *name*
:sh Retn	run shell, then return
:!*cmd* Retn	run *cmd*, then return
:n Retn	edit next file in arglist
:n *args* Retn	specify new arglist
^G	show current file and line
:ta *tag* Retn	to tag file entry *tag*
^]	:ta, following word is *tag*

In general, any **ex** or **ed** command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a Retn.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end of file is the default)
/*pat*	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/*pat*/+*n*	*n*th line after *pat*
?*pat*?-*n*	*n*th line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching () { or }

Adjusting the screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z-CR	... at bottom
z.CR	... at center
/*pat*/z-CR	*pat* line at bottom
z*n*.CR	use *n* line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

``	move cursor to previous context
'' ...	at first non-white in line
m*x*	mark current position with letter *x*
`*x*	move cursor to mark *x*
'*x* ...	at first non-white in line

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
-	previous line, at first non-white
CR	return, same as +
or j	next line, same column
or k	previous line, same column

Character positioning

^	first non-white
0	beginning of line
$	end of line
h or	forward
l or	backwards
^H	same as
space	same as
f*x*	find *x* forward
F*x*	f backward
t*x*	up to *x* forward
T*x*	back up to *x*
;	repeat last f F t or T
,	inverse of ;
|	to specified column
%	find matching ({) or }

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as ^H
kill	your kill, erase input this line
****	quotes ^H, your erase and kill
Esc	ends insertion, back to command
Del	interrupt, terminates insert
^D	backtab over *autoindent*
0^D	kill *autoindent*
^^D	same as 0^D, but restores indent next line
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before cursor
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text* Esc	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
=	indent for LISP
!	filter through command

Filter - !

Syntax: **!***cursor movement cmd* **RETN**

Function: Filters the text object delimited by the cursor and *cursor movement* through the XENIX command, *cmd*. For example, the following command sorts all lines between the cursor and the bottom of the screen, substituting the designated lines with the sorted lines:

!Lsort

Arguments and shell metacharacters may be included as part of *cmd*; however, standard input and output are always associated with the text object being filtered.

Miscellaneous Operations

C	change rest of line	**(c$)**
D	delete rest of line	**(d$)**
s	substitute chars	**(cl)**
S	substitute lines	**(cc)**
J	join lines	
x	delete characters	**(dl)**
X	... before cursor	**(dh)**
Y	yank lines	**(yy)**

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"*x***p**	put from buffer *x*
"*x***y**	yank to buffer *x*
"*x***d**	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d***p**	retrieve *d*'th last delete

Other Commands

The following command descriptions explain how to use miscellaneous **ex** commands.

abbr Maps the first argument to the following string. For example, the following command

:abbr rainbow yellow green blue red

maps "rainbow" to "yellow green blue red". Abbreviations can be turned off with the unabbreviate command, as in:

:una rainbow

map, map! Maps any character or escape sequence to an existing command sequence. Characters mapped with **map!** work only in insert mode, while characters mapped with **map** work only in command mode. The **unmap** command removes the mapping.

preserve The current editor buffer is saved as though the system had just crashed. Use this command only in emergencies when a w command has resulted in an error and you don't know how to save your work.

= Prints the line number of the addressed line. The current line is unchanged.

recover*file* Recovers *file* from the system save area. The system saves a copy of the editing buffer only if you have made changes to the *file*, the system crashes, or you execute a **preserve** command. Except when you use **preserve**, you will be notified by mail when a file is saved.

source*file* Reads and executes **ex** commands from the specified file. Source commands may be nested.

tag*label* The focus of editing switches to the location of label. If necessary, **vi** will switch to a different file in the current directory to find *label*. If you have modified the current file before giving a tag command, you must first write it out. If you give another tag command with no argument, the previous label is used.

Similarly, if you type only a Ctrl-], **vi** searches for the word immediately after the cursor as a tag. This is equivalent to typing ":tag", the word, and then a Retn.

The tags file is normally created by a program such as **ctags**, and consists of a number of lines with three fields separated by blanks or tabs. The first field gives the name of the tag, the second the name of the file where the tag resides, and the third gives an addressing form which can be used by the editor to find the tag. This field is usually a contextual scan using */pattern/* to be immune to minor changes

in the file. Such scans are always performed as if the **nomagic** option were set. The tag names in the tags file must be sorted alphabetically. There are a number of options that can be set to affect the **vi** environment. These can be set with the **ex** set command, either while editing or immediately after **vi** is invoked in the **vi** start-up file, .exrc.

version Prints the current version number of **vi.**

Each time **vi** is invoked, it reads commands from the file named .exrc in your home directory and from variable EXINIT in your environment. This file and variable normally set your preferred options so that they need not be set manually each time you invoke **vi.**

Options

autoprint ap **default: ap**

Causes the current line to be printed after each **ex** copy, move, or substitute command. This has the same effect as supplying a trailing "p" to each such command. **Autoprint** is suppressed in globals, and only applies to the last of many commands on a line.

beautify, bf **default: nobeautify**

Causes all control characters except tab, new line and formfeed to be discarded from the input. A complaint is registered the first time a backspace character is discarded. Beautify does not apply to command input.

directory, dir **default: dir=/tmp**

Specifies the directory in which **vi** places the editing buffer file. If this directory is not writeable, then the editor will exit abruptly when it fails to write to the buffer file.

edcompatible **default: noedcompatible**

Causes the presence or absence of g and c suffixes on substitute commands to be remembered, and to be toggled on and off by repeating the suffixes. The suffix **r** causes the substitution to be like the "~" command, instead of like &.

errorbells,eb **default: noeb**

Precedes error messages by a bell. If possible, the editor always places the error message in inverse video instead of ringing the bell.

hardtabs, ht **default: ht=8**

Gives the boundaries on which terminal hardware tabs are set, or on which the system expands tabs.

lisp **default: nolisp**

Indents appropriately for LISP code, and the ( ) { } [[and]] commands are modified to have meaning for LISP.

mesg **default: nomesg**

Causes write permission to be turned off to the terminal while you are in visual mode, if **nomesg** is set. This prevents people writing to your screen with the **write**(C) command and scrambling your screen as you edit.

open **default: open**

Doesn't permit the commands **open** and **visual** if set to **noopen.** This is set to prevent confusion resulting from accidental entry to open or visual mode.

optimize, opt **default: optimize**

Expedites output of text to the screen by setting the terminal so that it does not perform automatic carriage returns when printing more than one line of output, thus greatly speeding output on terminals without addressable cursors when text with leading whitespace is printed.

prompt **default: prompt**

Prompts **ex** input with a colon (:). If no prompt is set, when ex command mode is entered with the **Q** command, no colon prompt is displayed on the status line.

remap **default: remap**

Tries mapped characters repeatedly, if on, until they are unchanged. For example, if o is mapped to O and O is mapped to I, o will map to I if **remap** is set, and to O if noremap is set.

scroll **default: scroll=1/2 window**

Determines the number of logical lines scrolled when Ctrl-d is received from a terminal input in Command Mode, and the number of lines printed by a Command Mode **z** command (double the value of scroll).

shell, sh **default: sh=/bin/sh**

Gives the pathname of the shell forked for the shell escape command "!" , and by the shell command. The default is taken from SHELL in the environment, if present.

tabstop, ts **default: ts=8**

Expands tabs in the input file to be on tabstop boundaries for purposes of display.

taglength, tl **default: tl=0**

Makes the first taglength characters in a tag name significant, but all others are ignored. A value of zero (the default) means that all characters are significant.

tags **default: tags=tags /usr/lib/tags**

Uses a path of files as tag files for the tag command. A requested tag is searched for in the specified files, sequentially. By default files named **tags** are searched for in the current directory and in /usr/lib.

window **default: window=speed dependent**

Specifies the number of lines in a text window. The default is 8 at slow speeds (600 baud or less), 16 at medium speed (1200 baud), and the full screen (minus one line) at higher speeds.

w300, w1200, w9600

Sets the **window** option only if the speed is slow (300), medium (1200), or high (9600), respectively. These are not true options.

wrapmargin, wm **default: wm=0**

Defines the margin for automatic insertion of newlines during text input. A value of zero specifies no wrap margin.

writeany, wa **default: nowa**

Inhibits the checks normally made before write commands, allowing a write to any file that the system protection mechanism will allow.

Regular Expressions - New Read

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. **Vi** remembers two previous regular expressions: the previous regular expression used in a substitute command and the previous regular expression used elsewhere, referred to as the previous scanning regular expression. The previous regular expression can always be referred to by a null regular expression: e.g., "//" or "??".

The regular expressions allowed by **vi** are constructed in one of two ways depending on the setting of the **magic** option. The **ex** and **vi** default setting of **magic** gives quick access to a powerful set of regular expression metacharacters. The disadvantage of **magic** is that the user must remember that these metacharacters are magic and precede them with a backslash (\) to use them as "ordinary" characters. With **nomagic** set, regular expressions are much simpler, there being only two metacharacters. The power

of the other metacharacters is still available by preceding the now ordinary character with a "\". Note that "\" is thus always a metacharacter. In this discussion the **magic** option is assumed. With **nomagic**, the only special characters are the caret (^) at the beginning of a regular expression, the dollar sign ($) at the end of a regular expression, and the backslash (\). The tilde (~) and the ampersand (&) also lose their special meanings related to the replacement pattern of a substitute.

The following basic constructs are used to construct **magic** mode regular expressions.

char Matches an ordinary character with itself. Ordinary characters are any characters except a caret (^) at the beginning of a line, a dollar sign ($) at the end of line, a star (*) as any character other than the first, and any of the following characters:

.\ [~

These characters must be escaped (i.e., preceded) by a backslash (\) if they are to be treated as ordinary characters.

^ Forces, at the beginning of a pattern, the match to succeed only at the beginning of a line. $ Forces, at the end of a regular expression, the match to succeed only at the end of the line.

. Matches any single character except the newline character.

< Forces the match to occur only at the beginning of a "word"; that is, either at the beginning of a line, or just before a letter, digit, or underline and after a character that is not one of these.

> Matches the end of a "word"; similar to "\<", that is, either the end of the line or before a character that is not a letter, a digit, or the underline character.

[*string*] Matches any single character in the class defined by *string*. Most characters in *string* define themselves. A pair of characters separated by a dash (-) in *string* defines the set of characters between the specified lower and upper bounds. Thus, "[a-z]" as a regular expression matches any single lowercase letter. If the first character of string is a caret (^), the construct matches characters it otherwise would not. Thus, "[^a-z]" matches anything but a lowercase letter or a newline. To place a caret, left bracket, or dash in string, escape them with a preceding backslash (\).

The concatenation of two regular expressions first matches the leftmost regular expression and then the longest string that can be recognized as a regular expression. The first part of this new regular expression matches the first regular expression and the second part matches the second regular expression. Any of the single-character-matching regular expressions mentioned above may be followed by a "star" (*) to form a regular expression that matches zero or more adjacent occurrences of the characters matched by the prefixing regular expression. The tilde (~) may be used in a regular expression to match the text that defined the replacement part of the last s command. A regular expression may be enclosed between the sequences "\(" and "\)" to remember the text matched by the enclosed regular expression. This text can later be interpolated into the replacement text using the notation:

digit

where *digit* enumerates the set of remembered regular expressions.

The basic metacharacters for the replacement pattern are the ampersand (&) and the tilde (~). These are given as "\&" and "\~" when **nomagic** is set. Each instance of the ampersand is replaced by the characters matched by the regular expression. In the replacement pattern, the tilde stands for the text of the previous replacement pattern.

Other possible metasequences in the replacement pattern are always introduced by a backslash (/). The sequence "*n*" is 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. The sequences "\u" and "\l" cause the immediately following character in the replacement to be converted to uppercase or lowercase, respectively, if this character is a letter. The sequences "\U" and "L" turn such conversion on, either until "\E" or "\e" is encountered, or until the end of the replacement pattern.

Author

Vi and ex were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

Files

/usr/lib/terminfo/?/*	Compiled terminal description database

See Also

ed(C), edit(C), ex(C), and the *User's Guide*

Notes

Tampering with entries in /usr/lib/terminfo/?/* (for example, changing or removing an entry) can affect programs such as vi(C) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

Software tabs using ^T work only immediately after the *autoindent*. Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

Name

wait - Waits for completion of background processes.

Syntax

wait [*n*]

Description

This command waits until all background processes started with an ampersand (&) have finished and reports on abnormal terminations.

n Waits until the process with PID *n* exits, or no children exist.

Because the **wait**(S) system call must be executed in the parent process, the shell itself executes **wait** without creating a new process.

See Also

sh(C)

Notes

Not all of the processes of a pipeline with three or more stages are children of the shell, and cannot be waited for.

If *n* is not an active process id, all of your shell's currently active background processes are waited for and the return code will be zero.

Name

wall - Writes to all users.

Syntax

/etc/wall

Description

Wall reads its standard input until an end-of-file (Ctrl-d). It then sends the message you enter, preceded by the phrase "Broadcast Message," to all logged in users.

Log in as the super-user before executing wall to override any protections the users may have invoked. You can use **wall** to warn all users before shutting down the system.

Files

/dev/tty

Related Commands

mesg(C), write(C)

Diagnostics

"Cannot send to ..." when the open on a user's tty file fails.

Name

wc - Counts lines, words, and characters.

Syntax

wc [**-clw**] [*file ...*]

Description

Wc counts lines, words and characters in the named files, or in the standard input if no names appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or newlines. The options are:

-c Counts characters.

-l Counts lines.

-w Counts words.

The default is **-lwc**.

When files are specified on the command line, their names will be printed along with the counts.

Name

what - Identifies SCCS files.

Syntax

what [-s] *file...*

Description

What searches the given files for all occurrences of the pattern that **get**(C) substitutes for %Z% (this is **@(#)** at this printing) and prints out what follows until the first ", >, newline, \, or null character. For example, if the C program in file **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**(C), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

-s Quit after finding the first occurrence of pattern in each file.

See Also

get(C), help(C)

Diagnostics

Exit status is 0 if any matches are found, otherwise 1.

Notes

It is possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

Name

whereis - Locates source, binary, or manual for program.

Syntax

whereis [**-bms**] [**-u**] [**-BMS** *dir ...* **-f**] *name ...*

Description

Whereis locates source, binary and manual sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". Prefixes of "s." resulting from use of source code control are also dealt with. **Whereis** then attempts to locate the desired program in a list of standard places.

-b Searches for binary sections only.

-m Searches for manual sections only.

-s Searches for sources only.

-u Searches for unusual entries. A file is said to be unusual if it does not have one entry of each requested type.

For example,

whereis -m -u *

asks for those files in the current directory that have no documentation.

The **-B**, **-M**, and **-S** flags may be used to limit or otherwise change the places **whereis** searches. Each specifies a directory list in which to search for the corresponding type of file. The **-f** flag is used to terminate the last such directory list and signal the start of file names.

Example

The following finds all the files in /usr/bin that are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/bin
whereis -u -M /usr/man/man1 -S /usr/src/cmd -f *
```

Files

/usr/src/*
/usr/man/*
/lib, /etc, /bin
/usr/bin
/usr/lib

Notes

Since the program uses **chdir**(S) to run faster, pathnames given with the **-B**, **-M**, and **-S** must be full; i.e., they must begin with a "/".

Whereis does not use the environment variable PATH to locate files, but has a built-in list of directories to search. See **path**(M) for more information.

Name

who - Displays who is on the system.

Syntax

who [**-uT1Hqpdbrta**A**s**] [*file*]
who am i
who am I
who is on [*machine*] [**@**]

Description

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current system user. It examines the /etc/wtmp, which contains a history of all the logins since the file was last created.

Who with the **am i** or **am I** option identifies the invoking user.

The general format for output is:

name [*state*] *line time* [*idle*] [*pid*] [*comment*] [*exit*]

The *name*, *line*, and *time* information is produced by all options except **-q**; the *state* information is produced only by **-T**; the *idle* and *pid* information is produced only by **-u** and **-l**; and the *comment* and *exit* information is produced only by **-a**. The information produced for **-p**, **-d**, and **-r** is explained on the following pages.

With options, **who** can list logins, logouts, reboots, and changes to the system clock, as well as other processes spawned by the **init** process. These options are:

-u Lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory /dev. The *time* is the time that the user logged in. The *idle* column contains the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current."

If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old.** This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in /etc/inittab (see **inittab**(M)). This can contain information about where the terminal is located, the telphone number of the dataset, type of terminal if hardwired, etc.

-T Same as the -s option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a - appears if it is not. Root can write to all lines having a + or a - in the *state* field. If a bad line is encountered, a ? is printed.

-l Lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

-H Prints column headings above the regular output.

-q Displays only the names and the number users currently logged on (a quick **who**). When this option is used, all other options are ignored.

-p Lists any other process which is currently active and has been previously spawned by **init.** The *name* field is the name of the program executed by **init** as found in /etc/inittab. The *state*, *line*, and *idle* fields have no meaning. The *comment* field shows the *id* field of the line from /etc/inittab that spawned this process. See **inittab**(M).

-d Displays all processes that have expired and not been respawned by **init.** The *exit* field appears for dead processes and contains the termination and exit values (as returned by **wait**(C)), of the dead process. This can be useful in determining why a process terminated.

-b Indicates the time and date of the last reboot.

-r Indicates the current *run-level* of the **init** process. In addition, it displays the number of times the system has been in the current run-level and the previous run-level.

-t Indicates the last change to the system clock (via the **date**(C) command) by root. See **su**(C).

-a Processes /etc/utmp or the named *file* with all options turned on.

-A Lists accounting information.

-s Lists only the *name*, *line*, and *time* fields (the default).

is on *machine*
Lists names of users on *machine* on a Worknet network.

is on @
Lists all users on all machines on a Worknet network.

Note to the super-user: after a shutdown to the single-user state, **who** returns a prompt; the reason is that since /etc/utmp is updated at login time and there is no login in single-user state, **who** cannot report accurately on this state. **Who am i**, however, returns the correct information.

Files

/etc/utmp
/etc/wtmp
/etc/inittab

See Also

date(C), login(C), mesg(C), su(C), init(M), wait(C), inittab(M), utmp(M)

Name

whoami - Prints current effective user id.

Syntax

whoami

Description

Whoami prints who you are. It prints the effective user id, even if you have run **su**(C). **Who am i** prints the real login, as it looks in /etc/utmp.

Files

/etc/passwd	Name data base

See Also

who(C), su(C)

Name

whom - Displays in a columnar format all logged in users.

Syntax

who [-l]

Description

The **whom** command displays the user ID, log-in time, and tty for all users currently logged in to the system.

This command is much like the **who** command, except that **whom** displays this information in a more convenient columnar farmat.

The -l option reports how long each user has been logged in instead of the log-in time.

See Also

who(C)

Name

write - Writes to another user.

Syntax

write *user* [*line*]

Description

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* (**tty***nn*) [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". 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 *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in /etc/utmp is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal...

Permission to **write** may be denied or granted by use of the **mesg**(C) command. Writing to others is normally allowed by default. Certain commands, such as **pr**(C), disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a writeinhibited terminal.

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.

Files

/etc/utmp	To find user
/bin/sh	To execute !

See Also

mail(C), mesg(C), pr(C), sh(C), who(C)

Diagnostics

user is not logged on
: The person you are trying to write to is not logged on.

Permission denied
: The person you are trying to write to denies that permission (with **mesg**).

Warning: cannot respond, use "!mesg -y"
: Your terminal is set to **mesg n** and the recipient cannot respond to you.

Can no longer write to user
: The recipient has denied permission (**mesg n**) after you had started writing.

Name

xargs - Constructs and executes commands.

Syntax

xargs [*flags*] [*command* [*initial-arguments*]]

Description

Xargs combines the fixed *initial-arguments* with arguments read from the standard input to execute the specified *command* one or more times. The number of arguments read for each command invocation and the manner in which they are combined are determined by the *flags* specified.

Command, which may be a shell file, is searched for using the shell $PATH variable. If *command* is omitted, /bin/echo is used.

Arguments read in from standard input are defined as contiguous strings of characters delimited by one or more spaces tabs, or newlines; empty lines are always discarded. Spaces and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotation marks (single or double) are taken literally, and the delimiting quotation marks are removed. Outside of quoted strings, a backslash (\) will escape the next character.

Construct each argument list starting with the *initial-arguments*, followed by some number of arguments read from standard input (exception: see **-i** flag). The **-i**, **-l**, and **-n** flags determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then command is executed with the accumulated args. This process is repeated until there are no more arguments. When there are *flag* conflicts (e.g., **-l** vs. **-n**), the last flag has precedence. *Flag* values are:

-l*number* Command is executed for each *number* lines of nonempty arguments from the standard input. This is instead of the default single line of input for each command. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first newline unless the last character of the line is a space or a tab; a trailing space/tab signals continuation through the next nonempty line. If *number* is omitted, 1 is assumed. The -x option is forced.

-i*replstr* Insert mode: *command* is executed for each line from the standard input, taking the entire line as a single arg, and inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of five arguments in *initial-arguments* may each contain one or more instances of *replstr*. Spaces and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and the -x option is also forced. {} is assumed for *replstr* if not specified.

-n*number* Executes *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If the -x option is also coded, each *number* arguments must fit in the size limitation, or else **xargs** terminates execution.

-s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If -s is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

-p Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**-t**) is turned on to print the command to be executed, followed by a ?... prompt. A reply of "y" (optionally followed by anything) will execute the command; anything else, including just a Retn, skips that particular invocation of *command*.

-x Causes **xargs** to terminate if any argument list would be greater than *size* characters; **-x** is forced by the **-i** and **-l** options. When none of the **-i**, **-l**, or **-n** options are coded, the total length of all arguments must be within the *size* limit.

-e*eofstr* *Eofstr* is taken as the logical end-of-file string. Underscore (_) is assumed for the logical EOF string if **-e** is not coded. **-e** with no *eofstr* coded turns off the logical EOF string capability (underscore is taken literally). **Xargs** reads standard input until either end-of-file or the logical EOF string is encountered.

Xargs terminates if it either receives a return code of -1 from *command*, or if it cannot execute *command*. When *command* is a shell program, it should explicitly exit (see **sh**(C)) with an appropriate value to avoid returning accidentally with -1.

Examples

The following example will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of the file named log:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be printed and prints them one at a time:

```
ls | xargs -p -1 lpr
```

Or many at a time:

```
ls | xargs -p -1 | xargs lpr
```

The following example will execute diff(C) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

See Also

sh(C)

Name

xtty - Sets the options for a terminal.

Syntax

xtty [**-a**] [**-g**] [*options*] [tty*nn*] [*device ...*]
modem tty*nn*
unmodem tty*nn*

Description

Xtty sets the options for a terminal. It is an extension of the **stty**(C) command. To list settings, type **xtty** with no options.

There are three ways ports are used that affect modems: as a local port, as a dial-in port, and as a dial-out port.

On normal asynchronous communication ports, there is one output line (data-set-ready) and one input line (data-terminal-ready). Data-set-ready controls output for local ports and carrier present for modems. Data-terminal-ready controls input for local ports and carrier present for modems.

Options

-a Reports all option settings.

-g Reports current settings that can be used as an argument to another **xtty** command.

disable Turns off (disables) the specified terminal(s). For example, to turn off tty01 and tty02, type **xtty disable tty01 tty02.** You can disable and enable (see below) terminals in a single command. For example, type **xtty disable tty01 -e tty02** to disable tty01 and enable tty02.

enable Turns on (enables) the specified terminal(s). For example, to turn on tty01 and tty02, type **xtty enable tty01 tty02.** You can enable and disable (see above) terminals in a single command. For example, type **xtty enable tty01 -d tty02** to enable tty01 and disable tty02.

[-]iflow Turns on hardware input flow control. The minus sign (-) turns it off.

[-]oflow Turns on hardware output flow control. The minus sign (-) turns it off.

modem Sets up the specified tty port for use with a modem. For example, to set up tty01, type **xtty modem tty01.**

unmodem Unsets a tty port for modem use -- the reverse of modem (see above).

tty*nn* Specifies the port, where *nn* is a 2-digit port number (e.g., 05). If no port is specified, standard input is used.

/dev/*xxx* Specifies a device, where *xxx* is the device (e.g., console). If no port is specified, standard input is used.

To use a port as a local port, enable it and set the **modem** flag to either **user** or **off** (see **modem**). Use your own discretion to set **hupcl** (if **modem** is **user**), **iflow**, and **oflow**. Normally, **hupcl** is true, **iflow** is false, and **oflow** is true.

To use a port as a dial-in port, enable it and set the **modem** flag to **on** and **hupcl** to true. **Iflow**, **oflow**, and **clocal** should be false.

To use a port as a dial-out port, disable it. Set the **modem** flag to **user** and **hupcl** to true. **Iflow** and **oflow** should both be false. Set **clocal** at your own discretion. If you want to output to the modem when carrier is not present (e.g., to issue modem commands), open the port with the O_NDELAY flag set, or with **clocal** true.

All unknown flags are passed on to the **stty**(C) command. If a port was specified, it will be redirected to standard input.

Files

/dev/tty*nn*
/dev/console
/etc/ttys

See Also

setmodem(C), stty(C)

Name

yes - Prints a string repeatedly.

Syntax

yes [*string*]

Description

Yes repeatedly outputs "y", or if a single *string* argument is given, it is output repeatedly. The command will continue indefinitely unless aborted. Yes is useful in pipes to commands that prompt for input and require a "y" response for a yes. In this case, yes terminates when the command that it pipes to terminates, so that no infinite loop occurs.

Change Information

This is a summary of the changes that have been made to the previous version of this manual. The chapters, page numbers, and/or paragraphs mentioned in this summary reference the previous manual.

Title: Altos System V Series 386 Reference (C)

Revised Part Number: 690-22869-002

Previous Part Number: 690-22869-001

Date: June 1989

Changes:

Updated the Permuted Index and Table of Contents.

Added **shl**(C) and **whom**(C).

Renamed **ftp**(C) to **aftp**(C).

Changed the following pages:

Page	Command	Description
1	**acct**(C)	Removed reference to the **/etc/rc5** file, which no longer exists.
3	**at**(C)	Changed **queue** file name to **queuedefs.**
3	**cpio**(C)	Removed erroneous quaotation marks at end of example.
1	**cron**(C)	Changed file name **S75cron** to the more general form **S??cron.**

Page	Command	Description
2	**cron**(C)	Deleted information on **cron** error logging being turned on or off.
2	**ct**(C)	Added information on how **ct** interacts with existing **getty** or **uugetty** processes.
4	**cu**(C)	Data transfers with **cu** are initiated with the ~> characters, not the ~ character as previously documented.
2	**devinfo**(C)	Explained how SIO boards are reported as Mulidrops on 386 Series 1000 systems.
1	**devnm**(C)	Deleted references to **brc**(M).
4	**dos**(C)	Using "*.*" with **doscopy** copies all files from the root directory, not from the entire disk.
1,15	**ed**(C)	The -x encryption option is no longer offered.
1	**edit**(C)	The -x encryption option is no longer offered.
1	**enroll**(C)	Typing **?** displays valid **xget** commands.
1	**logname**(C)	The **logname** command does not get its value from $LOGNAME.
3	**lp**(C)	Described how to use **lp** to print a file that is not readable by others.
1	**setmode**(C)	Emphasized that **setmode** sets port modes for serial devices, not just serial printers.

Page	Command	Description
1	**sysconf**(C)	Described how SIO boards are reported on 386 Series 1000 systems.
1,4	**tar**(C)	Added **o** option and description for specifying the user and group IDs of extracted files.
2	**uux**(C)	Added description on how to use parentheses to enclose remote output files designations.
1	**vi**(C)	The **-x** encryption option is no longer offered.

(BLANK)

READER'S COMMENTS

Manual Title: Altos System V Series 386 Reference (C)

Part Number: 690-22869-002

Altos Computer Systems' Publications Department wants to provide documents that meet the needs of all our customers. Your comments help us produce better manuals.

Please Rate This Manual:	Excellent	Good	Average	Fair	Poor
Completeness of information	☐	☐	☐	☐	☐
Organization of manual	☐	☐	☐	☐	☐
Adequate illustrations	☐	☐	☐	☐	☐
Overall manual	☐	☐	☐	☐	☐

Do you find any of the chapters confusing or difficult to use? If so, which ones and why?

_____

_____

What could we do to improve the manual for you?

_____

_____

If you find errors or other problems when using this manual, please write them below. Do include page numbers or section titles.

_____

_____

Name: _____ Title: _____

Company: _____ Type of system: _____

Phone: (_____) _____ - _____ ext. _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7399 SAN JOSE, CA 95134

POSTAGE WILL BE PAID BY ADDRESSEE

Altos Computer Systems
ATTN: PUBLICATIONS DEPARTMENT
2641 Orchard Parkway
San Jose, CA 95134-9987
USA

Fold Here

P/N 690-22869-002
Printed in U.S.A.
9/89

Altos Computer Systems
2641 Orchard Parkway, San Jose, CA 95134
408/946-6700, FAX 408/433-9335