

UNIX is a Trade/Service Mark of the Bell System.

*This manual was set on a Graphic Systems, Inc. phototypesetter driven by the TROFF formatting program operating under the PWBIUNIX system. The text of the manual was prepared using the ED text editor.*

## ACKNOWLEDGEMENTS

The form and organization of this manual, as well as a major fraction of its contents, have been copied from the *UNIX Programmer's Manual—Sixth Edition*, by K. Thompson and D. M. Ritchie (Bell Telephone Laboratories, May 1975). The number of our colleagues who have contributed to UNIX and PWB/UNIX software and documentation is, by now, too large to list here, but the usefulness and acceptance of UNIX and of PWB/UNIX is a true measure of their collective success.

*Piscataway, New Jersey*  
*May 1977*

T.A.D.  
R.C.H.  
E.M.P.

## INTRODUCTION

This manual describes the features of PWB/UNIX. It provides neither a general overview of UNIX (for that, see "The UNIX Time-Sharing System," *Comm. ACM* 17(7):365-75, July 1974, by D. M. Ritchie and K. Thompson), nor details of the implementation of the system.

This manual is divided into eight sections:

- I. Commands and Application Programs
- II. System Calls
- III. Subroutines
- IV. Special Files
- V. File Formats and Conventions
- VI. Games
- VII. Miscellaneous
- VIII. System Maintenance

Section I (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory */bin* (for *binary* programs). Some programs also reside in */usr/bin*, to save space in */bin*. These directories are searched automatically by the command interpreter called the Shell.

Section II (*System Calls*) describes the entries into the UNIX supervisor, including the assembler and the C language interfaces. In the assembler, these system calls are invoked by the *sys* operation code, which is a synonym for the *trap* instruction.

Section III (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in directory */lib*. The subroutines available for C and for Fortran are also included there; they reside in */lib/libc.a* and */lib/libf.a*, respectively.

Section IV (*Special Files*) discusses the characteristics of each system "file" that actually refers to an input/output device. The names in that section refer to the Digital Equipment Corporation's device names for the hardware, instead of the names of the special files themselves.

Section V (*File Formats and Conventions*) documents the structure of particular kinds of files; for example, the form of the output of the assembler and the loader is given. Excluded are files used by only one command, for example, the assembler's intermediate files.

Section VIII (*System Maintenance*) discusses commands that are not intended for use by the ordinary user, in some cases because they disclose information in which he or she is presumably not interested, and in others because they perform privileged functions.

Each section consists of a number of independent entries of a page or so each. The name of the entry is in the upper corners of its pages. Entries within each section are alphabetized. The page numbers of each entry start at 1.

All entries are based on a common format, not all of whose parts always appear:

The **NAME** part repeats the name of the entry and states (very briefly) its purpose.

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section I (*Commands*):

**Boldface** strings are considered literals, and are to be typed just as they appear (they are usually underlined in the typed version of the manual entries *unless* they are juxtaposed with an *italic* string).

*Italic* strings usually represent substitutable arguments (they are underlined in the typed version of the manual entries).

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

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

A final convention is used by the commands themselves. An argument beginning with a minus sign “-” or a plus sign “+” is often taken to be some sort of flag argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with “-” or “+”.

The DESCRIPTION part discusses in detail the subject at hand.

The FILES part gives the file names that are built into the program.

The SEE ALSO part gives pointers to related information.

The DIAGNOSTICS part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The BUGS part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents (organized by section and alphabetized within each section) and a permuted index derived from that table precede Section I. Within each *index* entry, the title of the *manual* entry to which it refers is followed by the appropriate section number in parentheses. This fact is important because there is considerable name duplication among the sections, arising principally from commands that exist only to exercise a particular system call.

All manual entries are available on-line via the *man(I)* command (q.v.).

## HOW TO GET STARTED

This section provides the basic information you need to get started on UNIX (we will use "UNIX" in this section to mean both "UNIX" and "PWB/UNIX", unless the distinction matters): how to log in and log out, how to communicate through your terminal, and how to run a program. See *UNIX for Beginners* by B. W. Kernighan for a more complete introduction to the system.

*Logging in.* You must call UNIX from an appropriate terminal. UNIX supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained, together with the telephone number, from the system administrator. The same telephone number serves terminals operating at speeds of 110, 150, and 300 baud. After a data connection is established, the log in procedure depends on the kind of terminal you are using.

*300-baud terminals:* These terminals generally have a speed switch that should be set to "300" (or "30", for 30 characters per second) and a half-/full-duplex switch that should be set to full-duplex. When a connection is established, the system types "login:."; you type your user name, followed by the "return" key. If you have a password (and you should!), the system asks for it, but does not print ("echo") it on the terminal. After you have logged in, the "return", "new-line", and "line-feed" keys will give exactly the same result.

*Model 37 TELETYPE®:* When you have established a data connection, the system types out a few garbage characters (the "login:." message at the wrong speed). Depress the "break" (or "interrupt") key; this is a speed-independent signal to UNIX that a 150-baud terminal is in use. The system then will type "login:.", this time at 150 baud (another "break" at this point will get you down to 110 baud); you respond with your user name. From the Model 37 TELETYPE, and any other terminal that has the "new-line" function (combined "carriage-return" and "line-feed" pair), terminate each line you type with the "new-line" key (*not* the "return" key).

It is important that you type your name in lower case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and will translate all subsequent upper-case input to lower case. When you have logged in successfully, the Shell program will type a "%" to you. (The Shell is described below under *How to run a program.*)

For more information, consult *login(I)* and *getty(VIII)*, which discuss the login sequence in more detail, and *tty(IV)*, which discusses terminal input/output. See *terminals(VII)* for information about various terminals.

*Logging out.* There are three ways to log out:

You can simply hang up the phone.

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

You can also log in directly as another user by giving a *login* command.

*How to communicate through your terminal.* When you type to UNIX, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a "return" (or "new-line"), as described above in *Logging in.*

UNIX terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have the input characters interspersed. However, whatever you type will be saved and interpreted in correct sequence. There is a limit to the amount of read-ahead, but it

is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away *all* the saved characters.

On a terminal input line, the character “@” kills all the characters typed before it, so typing mistakes can be repaired on a single line. The character “#” erases the last character typed. Successive uses of “#” erase characters back to, but not beyond, the beginning of the line. “@” and “#” can be transmitted to a program by preceding them with “\”. (Thus, to erase “\”, you need two “#”s).

The ASCII “delete” (a.k.a. “rubout”) character is not passed to programs but instead generates an *interrupt signal*, just like the “break”, “interrupt”, or “attention” signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don’t want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed*(1), for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The *quit* signal is generated by typing the ASCII FS character. It not only causes a running program to terminate but also generates a file with the core image of the terminated process. *Quit* is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent about whether you have a terminal with the “new-line” function, or whether it must be simulated with a “carriage-return” and “line-feed” pair. In the latter case, all *input* “carriage-return” characters are changed to “line-feed” characters (the standard line delimiter), and a “carriage-return” and “line-feed” pair is echoed to the terminal. If you get into the wrong mode, the *stty*(1) command will rescue you.

Tab characters are used freely in UNIX source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty*(1) command will set or reset this mode. The system assumes that tabs are set every eight columns. The *tabs*(1) command will set tab stops on your terminal, if that is possible.

*How to run a program; the Shell.* When you have successfully logged into UNIX, a program called the Shell is listening to your terminal. The Shell reads typed-in lines, splits them up into a command name and arguments, and executes the command. A command is simply an executable program. Normally, the Shell looks first in your current directory (see *The current directory* below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the Shell can find them. The command name is always the first word on an input line to the Shell; it and its arguments are separated from one another by space or tab characters.

When a program terminates, the Shell will ordinarily regain control and type a “%” at you to indicate that it is ready for another command. The Shell has many other capabilities, which are described in detail in *sh*(1).

*The current directory.* UNIX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name). When you log in, any file name you type is by default assumed to be in this directory. Since you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their respective owners. As a matter of observed fact, many UNIX users do not protect their files from destruction, let alone perusal, by other users.

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

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

If your current directory has subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (without a prefixed “/”).

Without important exception, a path name may be used anywhere a file name is required.

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

For a fuller discussion of the file system, see “The UNIX Time-Sharing System” (*Comm. ACM* 17(7):365-75, July 1974) by D. M. Ritchie and K. Thompson. It may also be useful to glance through Section II of this manual, which discusses system calls, even if you don’t intend to deal with the system at that level.

*Writing a program.* To enter the text of a source program into a UNIX file, use *ed(I)*. The three principal languages available under UNIX are C (see *cc(I)*), Fortran (see *fc(I)*), and assembly language (see *as(I)*). After the program text has been entered through the editor and written in a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file in the current directory named “a.out”. (If the output is precious, use *mv(I)* to move it to a less exposed name soon.) If you wrote in assembly language, you will probably need to load the program with library subroutines; see *ld(I)*. The other two language processors call the loader automatically.

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

Next, you will need *cdb(I)* or *db(I)* to examine the remains of your program. The former is useful for C programs, the latter for assembly-language. No debugger is much help for Fortran.

Your programs can receive arguments from the command line just as system programs do. See *exec(II)*.

*Text processing.* Almost all text is entered through the editor *ed(I)*. The commands most often used to write text on a terminal are: *cat(I)*, *pr(I)*, and *nroff(I)*. The *cat(I)* command simply dumps ASCII text on the terminal, with no processing at all. The *pr(I)* command paginates the text, supplies headings, and has a facility for multi-column output. *Nroff(I)* is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file; it produces output on a typewriter-like terminal. *Roff(I)* is a less elaborate text formatting program, and requires somewhat less forethought; it is obsolescent. *Troff(I)* is similar to *nroff(I)*, but drives a Graphic Systems, Inc. phototypesetter. It was used to typeset this manual.

*Surprises.* Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you. To communicate with another user currently logged in, *write(I)* is used; *mail(I)* will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

When you log in, a message-of-the-day may greet you before the first “%”.

## TABLE OF CONTENTS

### I. COMMANDS AND APPLICATION PROGRAMS

450	handle special functions of DASI450 terminal
adb	debugger
admin	administer SCCS files
ar	archive and library maintainer
as	assembler
banner	print in block letters
bas	basic
bc	arbitrary precision interactive language
bdiff	big diff
bfs	big file scanner
cal	print calendar
cat	concatenate and print
cb	C beautifier
cc	C compiler
cd	change working directory
cdb	C debugger
chdir	change working directory
chghist	change the history entry of an SCCS delta
chgrp	change group
chmod	change mode
chown	change owner
cmp	compare two files
col	filter reverse line feeds
comb	combine SCCS deltas
comm	print lines common to two files
cp	copy
cpio	copy file archives in and out
cpx	copy a file exactly
cref	make cross reference listing
crypt	encode/decode
csplit	context split
date	print and set the date
db	debug
dc	desk calculator
dd	convert and copy a file
delta	make an SCCS delta
deroff	remove nroff, troff, and eqn constructs
df	report disk free space
diff	differential file comparator
diff3	3-way differential file comparison
diffmark	mark changes between versions of a file
dsw	delete interactively
du	summarize disk usage
echo	echo arguments
ed	text editor
egrep	search a file for lines containing a pattern
eqn	typeset mathematics
= (equals)	shell assignment command
exit	terminate command file



expr	evaluate arguments as an algebraic expression
fc	Fortran compiler
fd2	redirect file descriptor 2 (diagnostic output)
fgrep	search a file for lines containing keywords
file	determine file type
find	find files
gath	gather real and virtual files
get	get generation from SCCS file
goto	command transfer
graph	draw a graph
grep	search a file for a pattern
gsi	handle special functions of GSI300 terminal
help	ask for help
hp	handle special functions of HP 2640 terminal
if	conditional command
kill	terminate a process
ld	link editor
lex	generate programs for simple lexical tasks
ln	make a link
login	sign onto UNIX
logname, logdir, logtty	information from login
ls	list contents of directory
m4	macro processor
mail	send mail to designated users
make	make a program
man	print on-line documentation
mesg	permit or deny messages
mkdir	make a directory
mm	run off document with PWB/MM
mv	move or rename a file
neqn	typeset mathematics on terminal
newgrp	log in to a new group
next	new standard input for shell procedure
nice	run a command at low priority
nm	print name list
nohup	run a command immune to hangups
nroff, troff	text formatters
od	octal dump
onintr	handle interrupts in shell files
passwd	change login password
plot: t300, t300s, t450	graphics filters
pr	print file
prof	display profile data
prt	print SCCS file
ps	process status
ptx	permuted in lex
pump	Shell data transfer command
pwd	working directory name
quiz	test your knowledge
rc	Ratfor compiler
reform	reformat text file
regcmp	regular expression compile
rgrep	search a file for a pattern

rjstat	RJE status and enquiries
rm	remove (unlink) files
rmdel	remove a delta from an SCCS file
rmdir	remove directory
roff	format text
rsh	restricted shell (command interpreter)
sccsdiff	compare two versions of an SCCS file
sed	stream editor
send	submit RJE job
sh	shell (command interpreter)
shift	adjust Shell arguments
size	size of an object file
sleep	suspend execution for an interval
sno	Snobol interpreter
sort	sort or merge files
spell	find spelling errors
spline	interpolate smooth curve
split	split a file into pieces
strip	remove symbols and relocation bits
stty	set terminal options
su	become privileged user
sum	print checksum of a file
switch	shell multi-way branch command
sync	update the super block
tabs	set tabs on terminal
tail	deliver the last part of a file
tbl	format tables for nroff or troff
tee	pipe fitting
time	time a command
tp	manipulate DECTape and magtape
tr	transliterate
troff	text formatter
tty	get terminal name
typo	find possible typos
uname	print name of current UNIX
uniq	report repeated lines in a file
units	conversion program
vp	Versatec print
wait	await completion of process
wc	word count
what	identify files
whatsnew	compare file modification dates
while	shell iteration command
who	who is on the system
write	write to another user
xargs	construct argument list(s) and execute command
yacc	yet another compiler-compiler

## II. SYSTEM CALLS

intro	introduction to system calls
access	determine accessibility of file
alarm	schedule signal after specified time

break, brk, sbrk	change core allocation
chdir	change working directory
chmod	change mode of file
chown	change owner and group of a file
close	close a file
creat	create a new file
csw	read console switches
dup	duplicate an open file descriptor
exec, execl, execv	execute a file
exit	terminate process
fork	spawn new process
fstat	get status of open file
getgid	get group identifications
getpid	get process identification
getuid	get user identifications
gtty	get terminal status
indir	indirect system call
kill	send signal to a process
link	link to a file
logname, logdir, logtty, logpost	login information
mknod	make a directory or a special file
mount	mount file system
nice	set program priority
open	open for reading or writing
pause	indefinite wait
pipe	create an interprocess channel
profil	execution time profile
ptrace	process trace
read	read from file
seek	move read/write pointer
setgid	set process group ID
setpgrp	set process group number
setuid	set process user ID
signal	catch or ignore signals
stat	get file status
stime	set time
stty	set mode of terminal
sync	update super-block
tell	get file offset
time	get date and time
times	get process times
udata	get per-user data
umount	dismount file system
uname	get name of current PWB/UNIX
unlink	remove directory entry
ustat	get file system statistics
utime	update times in file
wait	wait for process to terminate
write	write on a file

### III. SUBROUTINES

abort	generate an IOT fault
abs, fabs	absolute value
alloc, free	core allocator
atan, atan2	arc tangent function
atof	convert ASCII to floating
atoi	convert ASCII to integer
cgetpid	return character form of process ID
crypt	password encoding
ctime, localtime, gmtime	convert date and time to ASCII
descend	search UNIX file system directories
ecvt, fcvt	output conversion
end, etext, edata	last locations in program
exp	exponential function
floor, ceil	floor and ceiling functions
fmod	floating modulo function
fptrap	floating point interpreter
gamma	log gamma function
getarg, iargc	get command arguments from Fortran
getc, getw, fopen	buffered input
getchar	read character
getpw	get name from UID
gmatch	match a string with a pattern (like glob(VIII))
hmul	high-order product
ierror	catch Fortran errors
ldiv, lrem	long division
locv	long output conversion
log	natural logarithm
monitor	prepare execution profile
nargs	argument count
nlist	get entries from name list
perror, sys_errlist, sys_nerr, errno	system messages
pexec	path search and execute a file
plot: openpl et al.	graphics interface
pow	floating exponentiation
printf	formatted print
putc, putw, fcreat, fflush	buffered output
putchar, flush	write character
qsort	quicker sort
rand, srand	random number generator
regcmp, regex	compile and execute regular expressions
reset, setexit	execute non-local goto
setfil	specify Fortran file name
setjmp, longjmp	execute non-local goto
sin, cos	trigonometric functions
sleep	suspend execution for interval
sqrt	square root function
strcpy, strcat, strcmp, strlen	operations on ASCII strings
ttyn	return name of current terminal

#### IV. SPECIAL FILES

cat	phototypesetter interface
dh	DH-11 communications multiplexer
dn	DN-11 ACU interface
dp	DP-11, DU-11 synchronous line interface
hp	RP04/RP05/RP06 moving-head disk
hs	RS03/RS04 fixed-head disk
ht	TU16 magtape interface
kl	KL-11 or DL-11 asynchronous interface
lp	line printer
mem, kmem, null	core memory
rje	DQS-11B interface for remote job entry
rp	RP-11/RP03 moving-head disk
tm	TM11/TU10 magtape interface
tty	general terminal interface

#### V. FILE FORMATS AND CONVENTIONS

a.out	assembler and link editor output
ar	archive (library) file format
ascii	map of ASCII character set
checklist	list of file systems processed by check
core	format of core image file
cpio	format of cpio archive
directory	format of directories
dump	incremental dump tape format
ebcdic	file format
fs	format of file system volume
fspec	format specification in text files
greek	graphics for extended TELETYPE Model 37 type-box
group	group file
master	master device information table
mnttab	mounted file system table
passwd	password file
plot	graphics interface
sccsfile	format of SCCS file
sha	Shell accounting file
tp	mag tape format
ttys	terminal initialization data
utmp	user information
wtmp	user login history

#### VI. GAMES

azel	satellite predictions
bio	biorhythm analysis
bj	the game of black jack
chess	the game of chess
cubic	three dimensional tic-tac-toe
factor	discover prime factors of a number
moo	guessing game
othello	a game of dramatic reversals
sky	obtain ephemerides

ttt . . . . . the game of tic-tac-toe  
wump . . . . . the game of hunt-the-wump

VII. MISCELLANEOUS

terminals . . . . . descriptions of commonly-used terminals  
DASI450 . . . . . DASI450, DIABLO 1620, XEROX 1700 terminals  
GSI300 . . . . . GSI300 (DTC300 or DASI300) hard-copy terminals  
HP2640 . . . . . Hewlett-Packard 2640 CRT terminal family  
TermiNet . . . . . GE TermiNet 300 (and 1200) terminals  
TI700 . . . . . TI 745, 735, and 725 terminals  
tmac.name . . . . . standard nroff and troff macro packages

VIII. SYSTEM MAINTENANCE

70boot . . . . . 11/70 bootstrap procedures  
ac . . . . . login accounting  
bcopy . . . . . disk block copy  
check . . . . . file system consistency check  
clri . . . . . clear i-node  
clrm . . . . . clear mode of i-node  
config . . . . . configure a system  
crash . . . . . what to do when the system crashes  
cron . . . . . clock daemon  
cu . . . . . call UNIX  
dcat . . . . . read/write synchronous line  
dcheck . . . . . file system directory consistency check  
devnm . . . . . device name  
diskboot . . . . . disk bootstrap programs  
dump . . . . . incremental file system dump  
fsdb . . . . . file system debugger  
getty . . . . . set terminal mode  
glob . . . . . generate command arguments  
hasp . . . . . PWB/UNIX IBM Remote Job Entry  
icheck . . . . . file system storage consistency check  
init . . . . . process control initialization  
lastcom . . . . . search shell accounting records  
mkfs . . . . . construct a file system  
mknod . . . . . build special file  
mount . . . . . mount file system  
ncheck . . . . . generate names from i-numbers  
patchup . . . . . patch up a damaged file system  
regen . . . . . regenerate system directories  
restor . . . . . incremental file system restore  
rmall . . . . . remove all  
romboot . . . . . special ROM bootstrap loaders  
sa . . . . . Shell accounting  
setmnt . . . . . establish mnttab table  
setuid . . . . . set user id of command  
shutdown . . . . . terminate all processing  
tapeboot . . . . . magnetic tape bootstrap programs  
umount . . . . . dismount file system  
unixboot . . . . . UNIX startup and boot procedures

volcopy, labelit . . . . . copy filesystems with label checking  
wall . . . . . write to all users

## PERMUTED INDEX

	= (equals)(I): shell assignment command
70boot(VIII):	11/70 bootstrap procedures
TermiNet(VII): GE TermiNet 300 (and	1200) terminals
DASI450(VII): DASI450, DIABLO	1620, XEROX 1700 terminals
DASI450, DIABLO 1620, XEROX	1700 terminals...DASI450(VIII):
fd2(I): redirect file descriptor	2 (diagnostic output)
HP2640(VII): Hewlett-Packard	2640 CRT terminal family
hp(I): handle special functions of HP	2640 terminal
TermiNet(VII): GE TermiNet	300 (and 1200) terminals
graphics for extended TELETYPE Model	37 type-box...greek(V):
diff3(I):	3-way differential file comparison
	450(I): handle special functions of DASI450 terminal
	70boot(VIII): 11/70 bootstrap procedures
TI700(VII): TI 745, 735, and	725 terminals
TI700(VII): TI 745,	735, and 725 terminals
TI700(VII): TI	745, 735, and 725 terminals
	abort(III): generate an IOT fault
	abs, fabs(III): absolute value
abs, fabs(III):	absolute value
access(II): determine	accessibility of file
	access(II): determine accessibility of file
sha(V): Shell	accounting file
lastcom(VIII): search shell	accounting records
ac(VIII): login	accounting
sa(VIII): Shell	accounting
dn(IV): DN-11	ACU interface
	ac(VIII): login accounting
	adb(I): debugger
shift(I):	adjust Shell arguments
admin(I):	administer SCCS files
alarm(II): schedule signal	administer SCCS files
	after specified time
expr(I): evaluate arguments as an	alarm(II): schedule signal after specified time
plot: openpl et	algebraic expression
	al.(III): graphics interface
break, brk, sbrk(II): change core	alloc, free(III): core allocator
alloc, free(III): core	allocation
rmall(VIII): remove	allocator
bio(VI): biorhythm	all
TermiNet(VII): GE TermiNet 300	analysis
yacc(I): yet	(and 1200) terminals
write(I): write to	another compiler-compiler
	another user
bc(I):	a.out(V): assembler and link editor output
atan, atan2(III):	arbitrary precision interactive language
ar(I):	arc tangent function
ar(V):	archive and library maintainer
cpio(V): format of cpio	archive (library) file format
cpio(I): copy file	archive
nargs(III):	archives in and out
	argument count



xargs(I): construct	argument list(s) and execute command
expr(I): evaluate	arguments as an algebraic expression
getarg, largc(III): get command	arguments from Fortran
echo(I): echo	arguments
glob(VIII): generate command	arguments
shift(I): adjust Shell	arguments
	ar(I): archive and library maintainer
	ar(V): archive (library) file format
ascii(V): map of	ASCII character set
strcat, strcmp, strlen(III): operations on	ASCII strings...strcpy,
atof(III): convert	ASCII to floating
atoi(III): convert	ASCII to integer
gmtime(III): convert date and time to	ASCII...ctime, localtime,
	ascii(V): map of ASCII character set
	as(I): assembler
help(I):	ask for help
a.out(V):	assembler and link editor output
as(I):	assembler
= (equals) (I): shell	assignment command
kl(IV): KL-11 or DL-11	asynchronous interface
	atan, atan2(III): arc tangent function
atan,	atan2(III): arc tangent function
	atof(III): convert ASCII to floating
	atoi(III): convert ASCII to integer
wait(I):	await completion of process
	azel(VI): satellite predictions
	banner(I): print in block letters
	bas(I): basic
bas(I):	basic
	bc(I): arbitrary precision interactive language
	bcopy(VIII): disk block copy
	bdiff(I): big diff
cb(I): C	beautifier
su(I):	become privileged user
diffmark(I): mark changes	between versions of a file
	bfs(I): big file scanner
bdiff(I):	big diff
bfs(I):	big file scanner
bio(VI):	biorhythm analysis
	bio(VI): biorhythm analysis
strip(I): remove symbols and relocation	bits
	bj(VI): the game of black jack
bj(VI): the game of	black jack
bcopy(VIII): disk	block copy
banner(I): print in	block letters
sync(I): update the super	block
unixboot(VIII): UNIX startup and	boot procedures
romboot(VIII): special ROM	bootstrap loaders
70boot(VIII): 11/70	bootstrap procedures
diskboot(VIII): disk	bootstrap programs
tapeboot(VIII): magnetic tape	bootstrap programs
switch(I): shell multi-way	branch command
	break, brk, sbrk(II): change core allocation

break,	brk, sbrk(II):	change core allocation
getc, getw, fopen(III):		buffered input
putc, putw, fcreat, fflush(III):		buffered output
mknod(VIII):		build special file
list of file systems processed	by check...checklist(V):	
cb(I):		C beautifier
cc(I):		C compiler
cdb(I):		C debugger
dc(I):	desk	calculator
cal(I):	print	calendar
		cal(I): print calendar
cu(VIII):		call UNIX
indir(II):	indirect system	call
intro(II):	introduction to system	calls
	ierror(III):	catch Fortran errors
	signal(II):	catch or ignore signals
		cat(I): concatenate and print
		cat(IV): phototypesetter interface
		cb(I): C beautifier
		cc(I): C compiler
		cdb(I): C debugger
		cd(I): change working directory
floor,	ceil(III):	floor and ceiling functions
floor, ceil(III):	floor and	ceiling functions
		cgetpid(III): return character form of process ID
break, brk, sbrk(II):		change core allocation
chgrp(I):		change group
passwd(I):		change login password
chmod(II):		change mode of file
chmod(I):		change mode
chown(II):		change owner and group of a file
chown(I):		change owner
chghist(I):		change the history entry of an SCCS delta
cd(I):		change working directory
chdir(I):		change working directory
chdir(II):		change working directory
diffmark(I):	mark	changes between versions of a file
pipe(II):	create an interprocess	channel
	cgetpid(III):	return character form of process ID
ascii(V):	map of ASCII	character set
getchar(III):	read	character
putchar, flush(III):	write	character
		chdir(I): change working directory
		chdir(II): change working directory
list of file systems processed by	check...checklist(V):	
check(VIII):	file system consistency	check
file system directory consistency	check...dcheck(VIII):	
file system storage consistency	check...icheck(VIII):	
labelit(VIII):	copy filesystems with label	checking...volcopy,
		checklist(V): list of file systems processed by check
sum(I):	print	checksum of a file
		check(VIII): file system consistency check
chess(VI):	the game of	chess

chess(VI): the game of chess  
 chghist(I): change the history entry of an SCCS delta  
 chgrp(I): change group  
 chmod(I): change mode  
 chmod(II): change mode of file  
 chown(I): change owner  
 chown(II): change owner and group of a file  
 ctri(VIII): clear i-node  
 clrm(VIII): clear mode of i-node  
 cron(VIII): clock daemon  
 close(II): close a file  
 close(II): close a file  
 ctri(VIII): clear i-node  
 clrm(VIII): clear mode of i-node  
 cmp(I): compare two files  
 col(I): filter reverse line feeds  
 comb(I): combine SCCS deltas  
 comb(I): combine SCCS deltas  
 getarg, iargc(III): get command arguments from Fortran  
 glob(VIII): generate command arguments  
 nice(I): run a command at low priority  
 = (equals) (I): shell assignment command  
 exit(I): terminate command file  
 nohup(I): run a command immune to hangups  
 rsh(I): restricted shell (command interpreter)  
 sh(I): shell (command interpreter)  
 goto(I): command transfer  
 if(I): conditional command  
 pump(I): Shell data transfer command  
 setuid(VIII): set user id of command  
 switch(I): shell multi-way branch command  
 time(I): time a command  
 while(I): shell iteration command  
 construct argument list(s) and execute command...xargs(I):  
 comm(I): print lines common to two files  
 comm(I): print lines common to two files  
 terminals(VII): descriptions of commonly-used terminals  
 dh(IV): DH-11 communications multiplexer  
 diff(I): differential file comparator  
 whatsnew(I): compare file modification dates  
 cmp(I): compare two files  
 sccsdiff(I): compare two versions of an SCCS file  
 diff3(I): 3-way differential file comparison  
 regcmp, regex(III): compile and execute regular expressions  
 cc(I): C compiler  
 yacc(I): yet another compiler-compiler  
 regcmp(I): regular expression compile  
 fc(I): Fortran compiler  
 rc(I): Ratfor compiler  
 wait(I): await completion of process  
 cat(I): concatenate and print  
 if(I): conditional command  
 config(VIII): configure a system

check(VIII): file system	config(VIII): configure a system
dcheck(VIII): file system directory	consistency check
icheck(VIII): file system storage	consistency check
csw(II): read	console switches
mkfs(VIII):	construct a file system
xargs(I):	construct argument list(s) and execute command
deroff(I): remove nroff, troff, and eqn	constructs
egrep(I): search a file for lines	containing a pattern
fgrep(I): search a file for lines	containing keywords
ls(I): list	contents of directory
csplit(I):	context split
init(VIII): process	control initialization
units(I):	conversion program
ecvt, fcvt(III): output	conversion
locv(III): long output	conversion
dd(I):	convert and copy a file
atof(III):	convert ASCII to floating
atoi(III):	convert ASCII to integer
ctime, localtime, gmtime(III):	convert date and time to ASCII
cpx(I):	copy a file exactly
dd(I): convert and	copy a file
cpio(I):	copy file archives in and out
volcopy, labelit(VIII):	copy filesystems with label checking
bcopy(VIII): disk block	copy
cp(I):	copy
break, brk, sbrk(II): change	core allocation
alloc, free(III):	core allocator
core(V): format of	core image file
mem, kmem, null(IV):	core memory
sin,	core(V): format of core image file
nargs(III): argument	cos(III): trigonometric functions
wc(I): word	count
cpio(V): format of	count
crash(VIII): what to do when the system	cp(I): copy
crash(VIII): what to do when the system crashes	cpio archive
creat(II):	cpio(I): copy file archives in and out
pipe(II):	cpio(V): format of cpio archive
creat(II): create a new file	cpx(I): copy a file exactly
cref(I): make cross reference listing	crashes
cron(VIII): clock daemon	crash(VIII): what to do when the system crashes
cref(I): make	create a new file
HP2640(VII): Hewlett-Packard 2640	create an interprocess channel
CRT terminal family	create a new file
crypt(I): encode/decode	cross reference listing
crypt(III): password encoding	cross reference listing
csplit(I): context split	CRT terminal family
csw(II): read console switches	crypt(I): encode/decode
ASCII...	crypt(III): password encoding
ctime, localtime, gmtime(III): convert date and time to	csplit(I): context split

uname(II): get name of	cubic(VI): three dimensional tic-tac-toe
ttyn(III): return name of	current PWB/UNIX
uname(I): print name of	current terminal
spline(I): interpolate smooth	current UNIX
	curve
	cu(VIII): call UNIX
cron(VIII): clock	daemon
patchup(VIII): patch up a	damaged file system
GSI300(VII): GSI300 (DTC300 or	DASI300) hard-copy terminals
DASI450(VII):	DASI450, DIABLO 1620, XEROX 1700 terminals
450(I): handle special functions of	DASI450 terminal
terminals...	DASI450(VII): DASI450, DIABLO 1620, XEROX 1700
prof(I): display profile	data
ttys(V): terminal initialization	data
udata(II): get per-user	data
ctime, localtime, gmtime(III): convert	date and time to ASCII
time(II): get	date and time
date(I): print and set the	date
	date(I): print and set the date
whatsnew(I): compare file modification	dates
	db(I): debug
	dcat(VIII): read/write synchronous line
	dcheck(VIII): file system directory consistency check
	dc(I): desk calculator
	dd(I): convert and copy a file
	db(I): debug
	adb(I): debugger
	cdb(I): C debugger
fsdb(VIII): file system	debugger
tp(I): manipulate	DECtape and magtape
dsw(I):	delete interactively
tail(I):	deliver the last part of a file
rm(II): remove a	delta from an SCCS file
change the history entry of an SCCS	delta...chghist(I):
delta(I): make an SCCS	delta
	delta(I): make an SCCS delta
comb(I): combine SCCS	deltas
mesg(I): permit or	deny messages
	deroff(I): remove nroff, troff, and eqn constructs
	descend(III): search UNIX file system directories
terminals(VII):	descriptions of commonly-used terminals
fd2(I): redirect file	descriptor 2 (diagnostic output)
dup(II): duplicate an open file	descriptor
mail(I): send mail to	designated users
	dc(I): desk calculator
	access(II): determine accessibility of file
	file(I): determine file type
master(V): master	device information table
devnm(VIII):	device name
	devnm(VIII): device name
	df(I): report disk free space
dh(IV):	DH-11 communications multiplexer
	dh(IV): DH-11 communications multiplexer

DASI450(VII): DASI450, DIABLO 1620, XEROX 1700 terminals  
 fd2(I): redirect file descriptor 2 (diagnostic output)  
 diff3(I): 3-way differential file comparison  
 bdiff(I): big diff  
 diff(I): differential file comparator  
 diff3(I): 3-way differential file comparison  
 diff(I): differential file comparator  
 diffmark(I): mark changes between versions of a file  
 cubic(VI): three dimensional tic-tac-toe  
 descend(III): search UNIX file system directories  
 directory(V): format of directories  
 regen(VIII): regenerate system directories  
 dcheck(VIII): file system directory consistency check  
 unlink(II): remove directory entry  
 pwd(I): working directory name  
 mknod(II): make a directory or a special file  
 cd(I): change working directory  
 chdir(I): change working directory  
 chdir(II): change working directory  
 ls(I): list contents of directory  
 mkdir(I): make a directory  
 rmdir(I): remove directory  
 directory(V): format of directories  
 factor(VI): discover prime factors of a number  
 bcopy(VIII): disk block copy  
 diskboot(VIII): disk bootstrap programs  
 df(I): report disk free space  
 du(I): summarize disk usage  
 diskboot(VIII): disk bootstrap programs  
 hp(IV): RP04/RP05/RP06 moving-head disk  
 hs(IV): RS03/RS04 fixed-head disk  
 rp(IV): RP-11/RP03 moving-head disk  
 umount(II): dismount file system  
 umount(VIII): dismount file system  
 prof(I): display profile data  
 ldiv, lrem(III): long division  
 kl(IV): KL-11 or DL-11 asynchronous interface  
 dn(IV): DN-11 ACU interface  
 dn(IV): DN-11 ACU interface  
 mm(I): run off document with PWB/MM  
 man(I): print on-line documentation  
 dp(IV): DP-11, DU-11 synchronous line interface  
 dp(IV): DP-11, DU-11 synchronous line interface  
 rje(IV): DQS-11B interface for remote job entry  
 othello(VI): a game of dramatic reversals  
 graph(I): draw a graph  
 dsw(I): delete interactively  
 GSI300(VII): GSI300 (DTC300 or DASI300) hard-copy terminals  
 dp(IV): DP-11, DU-11 synchronous line interface  
 du(I): summarize disk usage  
 dump(V): incremental dump tape format  
 dump(VIII): incremental file system dump  
 od(I): octal dump

dump(V): incremental dump tape format  
 dump(VIII): incremental file system dump  
 dup(II): duplicate an open file descriptor  
 dup(II): duplicate an open file descriptor  
 ebcdic(V): file format  
 echo(I): echo arguments  
 echo(I): echo arguments  
 ecvt, fcvt(III): output conversion  
 end, etext, edata(III): last locations in program  
 ed(I): text editor  
 editor output  
 a.out(V): assembler and link editor  
 ed(I): text editor  
 ld(I): link editor  
 sed(I): stream editor  
 egrep(I): search a file for lines containing a pattern  
 encode/decode  
 crypt(I): encode/decode  
 crypt(III): password encoding  
 end, etext, edata(III): last locations in program  
 enquiries  
 rjstat(I): RJE status and entries from name list  
 nlist(III): get entry of an SCCS delta  
 chghist(I): change the history Entry  
 hasp(VIII): PWB/UNIX IBM Remote Job entry  
 rje(IV): DQS-11B interface for remote job entry  
 unlink(II): remove directory entry  
 sky(VI): obtain ephemerides  
 deroff(I): remove nroff, troff, and eqn constructs  
 eqn(I): typeset mathematics  
 = (equals) (I): shell assignment command  
 perror, sys\_errlist, sys\_nerr, errno(III): system messages  
 ierror(III): catch Fortran errors  
 spell(I): find spelling errors  
 setmnt(VIII): establish mnttab table  
 plot: openpl et al. (III): graphics interface  
 end, etext, edata(III): last locations in program  
 expr(I): evaluate arguments as an algebraic expression  
 cpx(I): copy a file exactly  
 exec, execl, execv(II): execute a file  
 execl, execv(II): execute a file  
 exec, execl, execv(II): execute a file  
 pexec(III): path search and execute a file  
 xargs(I): construct argument list(s) and execute command  
 reset, setexit(III): execute non-local goto  
 setjmp, longjmp(III): execute non-local goto  
 regcmp, regex(III): compile and execute regular expressions  
 sleep(I): suspend execution for an interval  
 sleep(III): suspend execution for interval  
 monitor(III): prepare execution profile  
 profil(II): execution time profile  
 exec, execl, execv(II): execute a file  
 exit(I): terminate command file  
 exit(II): terminate process  
 exp(III): exponential function  
 exp(III): exponential function

pow(III): floating	exponentiation
regcmp(I): regular	expression compile
expr(I): evaluate arguments as an algebraic	expression
regex(III): compile and execute regular	expressions...regcmp,
	expr(I): evaluate arguments as an algebraic expression
greek(V): graphics for	extended TELETYPE Model 37 type-box
abs,	fabs(III): absolute value
factor(VI): discover prime	factors of a number
	factor(VI): discover prime factors of a number
Hewlett-Packard 2640 CRT terminal	family...HP2640(VII):
abort(III): generate an IOT	fault
	fc(I): Fortran compiler
putc, putw,	fcreat, fflush(III): buffered output
ecvt,	fcvt(III): output conversion
	fd2(I): redirect file descriptor 2 (diagnostic output)
col(I): filter reverse line	feeds
putc, putw, fcreat,	fflush(III): buffered output
	fgrep(I): search a file for lines containing keywords
cpio(I): copy	file archives in and out
diff(I): differential	file comparator
diff3(I): 3-way differential	file comparison
fd2(I): redirect	file descriptor 2 (diagnostic output)
dup(II): duplicate an open	file descriptor
cpx(I): copy a	file exactly
grep(I): search a	file for a pattern
rgrep(I): search a	file for a pattern
egrep(I): search a	file for lines containing a pattern
fgrep(I): search a	file for lines containing keywords
ar(V): archive (library)	file format
ebcdic(V):	file format
split(I): split a	file into pieces
whatsnew(I): compare	file modification dates
setfil(III): specify Fortran	file name
tell(II): get	file offset
bfs(I): big	file scanner
stat(II): get	file status
check(VIII):	file system consistency check
fsdb(VIII):	file system debugger
descend(III): search UNIX	file system directories
dcheck(VIII):	file system directory consistency check
dump(VIII): incremental	file system dump
restor(VIII): incremental	file system restore
ustat(II): get	file system statistics
icheck(VIII):	file system storage consistency check
mnttab(V): mounted	file system table
fs(V): format of	file system volume
mkfs(VIII): construct a	file system
mount(II): mount	file system
mount(VIII): mount	file system
patchup(VIII): patch up a damaged	file system
checklist(V): list of	file systems processed by check
umount(II): dismount	file system
umount(VIII): dismount	file system



file(I): determine	file type
access(II): determine accessibility of	file
chmod(II): change mode of	file
chown(II): change owner and group of a	file
close(II): close a	file
core(V): format of core image	file
creat(II): create a new	file
dd(I): convert and copy a	file
mark changes between versions of a	file...diffmark(I):
exec, execl, execv(II): execute a	file
exit(I): terminate command	file
fstat(II): get status of open	file
get(I): get generation from SCCS	file
group(V): group	file
	file(I): determine file type
link(II): link to a	file
mknod(II): make a directory or a special	file
mknod(VIII): build special	file
mv(I): move or rename a	file
passwd(V): password	file
pexec(III): path search and execute a	file
pr(I): print	file
prt(I): print SCCS	file
read(II): read from	file
reform(I): reformat text	file
rmddel(I): remove a delta from an SCCS	file
admin(I): administer SCCS	files
compare two versions of an SCCS	file...sccsdiff(I):
sccsfile(V): format of SCCS	file
cmp(I): compare two	files
comm(I): print lines common to two	files
find(I): find	files
fspec(V): format specification in text	files
gath(I): gather real and virtual	files
sha(V): Shell accounting	file
size(I): size of an object	file
onintr(I): handle interrupts in shell	files
rm(I): remove (unlink)	files
sort(I): sort or merge	files
sum(I): print checksum of a	file
what(I): identify	files
volcopy, labelit(VIII): copy	filesystems with label checking
tail(I): deliver the last part of a	file
uniq(I): report repeated lines in a	file
utime(II): update times in	file
write(II): write on a	file
col(I):	filter reverse line feeds
plot: t300, t300s, t450(I): graphics	filters
find(I):	find files
typo(I):	find possible typos
spell(I):	find spelling errors
find(I):	find files
tee(I): pipe	fitting

hs(IV): RS03/RS04	fixed-head disk
pow(III):	floating exponentiation
fmod(III):	floating modulo function
fptrap(III):	floating point interpreter
atof(III): convert ASCII to	floating
floor, ceil(III):	floor and ceiling functions
	floor, ceil(III): floor and ceiling functions
putchar,	flush(III): write character
	fmod(III): floating modulo function
getc, getw,	fopen(III): buffered input
	fork(II): spawn new process
cgetpid(III): return character	form of process ID
	core(V): format of core image file
	cpio(V): format of cpio archive
directory(V):	format of directories
	fs(V): format of file system volume
	sccsfile(V): format of SCCS file
	fspec(V): format specification in text files
	tbl(I): format tables for nroff or troff
	roff(I): format text
ar(V): archive (library) file	format
dump(V): incremental dump tape	format
ebcdic(V): file	format
printf(III):	formatted print
nroff, troff(I): text	formatters
troff(I): text	formatter
tp(V): mag tape	format
fc(I):	Fortran compiler
ierror(III): catch	Fortran errors
setfil(III): specify	Fortran file name
iargc(III): get command arguments from	Fortran...getarg,
	fptrap(III): floating point interpreter
df(I): report disk	free space
alloc,	free(III): core allocator
rm-del(I): remove a delta	from an SCCS file
read(II): read	from file
getarg, iargc(III): get command arguments	from Fortran
ncheck(VIII): generate names	from i-numbers
logname, logdir, logtty(I): information	from login
nlist(III): get entries	from name list
get(I): get generation	from SCCS file
getpw(III): get name	from UID
	fsdb(VIII): file system debugger
	fspec(V): format specification in text files
	fstat(II): get status of open file
	fs(V): format of file system volume
atan, atan2(III): arc tangent	function
exp(III): exponential	function
fmod(III): floating modulo	function
gamma(III): log gamma	function
450(I): handle special	functions of DASI450 terminal
gsi(I): handle special	functions of GSI300 terminal
hp(I): handle special	functions of HP 2640 terminal

floor, ceil(III): floor and ceiling functions  
 sqrt(III): square root function  
 sin, cos(III): trigonometric functions  
     bj(VI): the game of black jack  
     chess(VI): the game of chess  
     othello(VI): a game of dramatic reversals  
     wump(VI): the game of hunt-the-wumpus  
     ttt(VI): the game of tic-tac-toe  
 moo(VI): guessing game  
 gamma(III): log gamma function  
     gamma(III): log gamma function  
     gath(I): gather real and virtual files  
     gath(I): gather real and virtual files  
 TermiNet(VII): GE TermiNet 300 (and 1200) terminals  
     tty(IV): general terminal interface  
     abort(III): generate an IOT fault  
     glob(VIII): generate command arguments  
     ncheck(VIII): generate names from i-numbers  
     lex(I): generate programs for simple lexical tasks  
     get(I): get generation from SCCS file  
 rand, srand(III): random number generator  
     getarg, iargc(III): get command arguments from Fortran  
     time(II): get date and time  
     nlist(III): get entries from name list  
     tell(II): get file offset  
     stat(II): get file status  
     ustat(II): get file system statistics  
     get(I): get generation from SCCS file  
     getgid(II): get group identifications  
     getpw(III): get name from UID  
     uname(II): get name of current PWB/UNIX  
     udata(II): get per-user data  
     getpid(II): get process identification  
     times(II): get process times  
     fstat(II): get status of open file  
     tty(I): get terminal name  
     gtty(II): get terminal status  
     getuid(II): get user identifications  
     getarg, iargc(III): get command arguments from Fortran  
     getc, getw, fopen(III): buffered input  
     getchar(III): read character  
     getgid(II): get group identifications  
     get(I): get generation from SCCS file  
     getpid(II): get process identification  
     getpw(III): get name from U ID  
     getty(VIII): set terminal mode  
     getuid(II): get user identifications  
     getc, getw, fopen(III): buffered input  
     glob(VIII): generate command arguments  
     glob(VIII)...gmatch(III):  
 match a string with a pattern (like gmatch(III): match a string with a pattern (like  
     glob(VIII))... gmtime(III): convert date and time to ASCII  
     ctime, localtime, goto(I): command transfer

reset, setexit(III): execute non-local	goto
setjmp, longjmp(III): execute non-local	goto
graph(I): draw a	graph
	graph(I): draw a graph
plot: t300, t300s, t450(I):	graphics filters
greek(V):	graphics for extended TELETYPE Model 37 type-box
plot: openpl et al.(III):	graphics interface
plot(V):	graphics interface
type-box...	greek(V): graphics for extended TELETYPE Model 37
	grep(I): search a file for a pattern
group(V):	group file
getgid(II): get	group identifications
setgid(II): set process	group ID
setpgrp(II): set process	group number
chown(II): change owner and	group of a file
chgrp(I): change	group
newgrp(I): log in to a new	group
	group(V): group file
GSI300(VII):	GSI300 (DTC300 or DASI300) hard-copy terminals
gsi(I): handle special functions of	GSI300 terminal
terminals...	GSI300(VII): GSI300 (DTC300 or DASI300) hard-copy
	gsi(I): handle special functions of GSI300 terminal
	gtty(II): get terminal status
	moo(VI): guessing game
onintr(I):	handle interrupts in shell files
450(I):	handle special functions of DASI450 terminal
gsi(I):	handle special functions of GSI300 terminal
hp(I):	handle special functions of HP 2640 terminal
nohup(I): run a command immune to	hangups
GSI300(VII): GSI300 (DTC300 or DASI300)	hard-copy terminals
	hasp(VIII): PWB/UNIX IBM Remote Job Entry
help(I): ask for	help
	help(I): ask for help
HP2640(VII):	Hewlett-Packard 2640 CRT terminal family
hmul(III):	high-order product
chghist(I): change the	history entry of an SCCS delta
wtmp(V): user login	history
	hmul(III): high-order product
hp(I): handle special functions of	HP 2640 terminal
	HP2640(VII): Hewlett-Packard 2640 CRT terminal family
	hp(I): handle special functions of HP 2640 terminal
	hp(IV): RP04/RP05/RP06 moving-head disk
	hs(IV): RS03/RS04 fixed-head disk
	ht(IV): TU16 magtape interface
wump(VI): the game of	hunt-the-wumpus
getarg,	iargc(III): get command arguments from Fortran
hasp(VIII): PWB/UNIX	IBM Remote Job Entry
	icheck(VIII): file system storage consistency check
setuid(VIII): set user	id of command
return character form of process	ID...cgetpid(III):
getpid(II): get process	identification
getgid(II): get group	identifications
getuid(II): get user	identifications

what(I):	identify files
setgid(II): set process group	ID
setuid(II): set process user	ID
	ierror(III): catch Fortran errors
	if(I): conditional command
signal(II): catch or	ignore signals
core(V): format of core	image file
nohup(I): run a command	immune to hangups
dump(V):	incremental dump tape format
dump(VIII):	incremental file system dump
restor(VIII):	incremental file system restore
pause(II):	indefinite wait
ptx(I): permuted	index
indir(II):	indirect system call
	indir(II): indirect system call
logname, logdir, logtty(I):	information from login
master(V): master device	information table
logname, logdir, logtty, logpost(II):	login information
utmp(V): user	information
ttys(V): terminal	initialization data
init(VIII): process control	initialization
	init(VIII): process control initialization
clri(VIII): clear	i-node
clrm(VIII): clear mode of	i-node
next(I): new standard	input for shell procedure
getc, getw, fopen(III): buffered	input
atoi(III): convert ASCII to	integer
bc(I): arbitrary precision	interactive language
dsw(I): delete	interactively
rje(IV): DQS-11B	interface for remote job entry
cat(IV): phototypesetter	interface
dn(IV): DN-11 ACU	interface
dp(IV): DP-11, DU-11 synchronous line	interface
ht(IV): TU16 magtape	interface
kl(IV): KL-11 or DL-11 asynchronous	interface
plot: openpl et al. (III): graphics	interface
plot(V): graphics	interface
tm(IV): TM11/TU10 magtape	interface
tty(IV): general terminal	interface
spline(I):	interpolate smooth curve
fptrap(III): floating point	interpreter
rsh(I): restricted shell (command	interpreter)
sh(I): shell (command	interpreter)
sno(I): Snobol	interpreter
pipe(II): create an	interprocess channel
onintr(I): handle	interrupts in shell files
sleep(I): suspend execution for an	interval
sleep(III): suspend execution for	interval
intro(II):	introduction to system calls
	intro(II): introduction to system calls
ncheck(VIII): generate names from	i-numbers
abort(III): generate an	IOT fault
while(I): shell	iteration command

bj(VI): the game of black	jack
hasp(VIII): PWB/UNIX IBM Remote	Job Entry
rje(IV): DQS-11B interface for remote	job entry
send(I): submit RJE	job
search a file for lines containing	keywords...fgrep(I):
	kill(I): terminate a process
	kill(II): send signal to a process
kl(IV):	KL-11 or DL-11 asynchronous interface
	kl(IV): KL-11 or DL-11 asynchronous interface
mem,	kmem, null(IV): core memory
quiz(I): test your	knowledge
labelit(VIII): copy filesystems with	label checking...volcopy,
volcopy,	labelit(VIII): copy filesystems with label checking
bc(I): arbitrary precision interactive	language
end, etext, edata(III):	last locations in program
tail(I): deliver the	last part of a file
	lastcom(VIII): search shell accounting records
	ld(I): link editor
	ldiv, lrem(III): long division
banner(I): print in block	letters
	lex(I): generate programs for simple lexical tasks
lex(I): generate programs for simple	lexical tasks
ar(V): archive	(library) file format
ar(I): archive and	library maintainer
gmatch(III): match a string with a pattern	(like glob(VIII))
col(I): filter reverse	line feeds
dp(IV): DP-11, DU-11 synchronous	line interface
lp(IV):	line printer
dcat(VIII): read/write synchronous	line
comm(I): print	lines common to two files
egrep(I): search a file for	lines containing a pattern
fgrep(I): search a file for	lines containing keywords
uniq(I): report repeated	lines in a file
a.out(V): assembler and	link editor output
ld(I):	link editor
link(II):	link to a file
	link(II): link to a file
ln(I): make a	link
ls(I):	list contents of directory
checklist(V):	list of file systems processed by check
cref(I): make cross reference	listing
nlist(III): get entries from name	list
nm(I): print name	list
xargs(I): construct argument	list(s) and execute command
	ln(I): make a link
romboot(VIII): special ROM bootstrap	loaders
ctime,	localtime, gmtime(III): convert date and time to ASCII
end, etext, edata(III): last	locations in program
	locv(III): long output conversion
gamma(III):	log gamma function
newgrp(I):	log in to a new group
log(III): natural	logarithm
logname,	logdir, logtty, logpost(II): login information

logname,	logdir, logtty(I): information from login
	log(III): natural logarithm
ac(VIII):	login accounting
wtmp(V): user	login history
logname, logdir, logtty, logpost(II):	login information
passwd(I): change	login password
	login(I): sign onto UNIX
logdir, logtty(I): information from	login...logname,
	logname, logdir, logtty, logpost(II): login information
	logname, logdir, logtty(I): information from login
logname, logdir, logtty,	logpost(II): login information
logname, logdir,	logtty, logpost(II): login information
logname, logdir,	logtty(I): information from login
ldiv, lrem(III):	long division
locv(III):	long output conversion
setjmp,	longjmp(III): execute non-local goto
nice(I): run a command at	low priority
	lp(IV): line printer
	ldiv, lrem(III): long division
	ls(I): list contents of directory
	m4(I): macro processor
tmac.name(VII): standard nroff and troff	macro packages
	m4(I): macro processor
	tp(V): mag tape format
tapeboot(VIII):	magnetic tape bootstrap programs
ht(IV): TU16	magtape interface
tm(IV): TM11/TU10	magtape interface
tp(I): manipulate DECTape and	magtape
mail(I): send	mail to designated users
	mail(I): send mail to designated users
ar(I): archive and library	maintainer
mknod(II):	make a directory or a special file
mkdir(I):	make a directory
ln(I):	make a link
make(I):	make a program
delta(I):	make an SCCS delta
cref(I):	make cross reference listing
	make(I): make a program
	man(I): print on-line documentation
tp(I):	manipulate DECTape and magtape
ascii(V):	map of ASCII character set
diffmark(I):	mark changes between versions of a file
master(V):	master device information table
	master(V): master device information table
gmatch(III):	match a string with a pattern (like glob(VIII))
neqn(I): typeset	mathematics on terminal
eqn(I): typeset	mathematics
	mem, kmem, null(IV): core memory
mem, kmem, null(IV): core	memory
sort(I): sort or	merge files
	mesg(I): permit or deny messages
mesg(I): permit or deny	messages
sys_errlist, sys_nerr, errno(III): system	messages...perror,

	mkdir(I): make a directory
	mkfs(VIII): construct a file system
	mknod(II): make a directory or a special file
	mknod(VIII): build special file
	mm(I): run off document with PWB/MM
setmnt(VIII): establish	mnttab table
	mnttab(V): mounted file system table
chmod(II): change	mode of file
chrm(VIII): clear	mode of i-node
stty(II): set	mode of terminal
chmod(I): change	mode
getty(VIII): set terminal	mode
greek(V): graphics for extended TELETYPE	Model 37 type-box
whatsnew(I): compare file	modification dates
fmod(III): floating	modulo function
	monitor(III): prepare execution profile
	moo(VI): guessing game
mount(II):	mount file system
mount(VIII):	mount file system
mnttab(V):	mounted file system table
	mount(II): mount file system
	mount(VIII): mount file system
mv(I):	move or rename a file
seek(II):	move read/write pointer
hp(IV): RP04/RP05/RP06	moving-head disk
rp(IV): RP-11/RP03	moving-head disk
dh(IV): DH-11 communications	multiplexer
switch(I): shell	multi-way branch command
	mv(I): move or rename a file
getpw(III): get	name from UID
nlist(III): get entries from	name list
nm(I): print	name list
uname(II): get	name of current PWB/UNIX
ttyn(III): return	name of current terminal
uname(I): print	name of current UNIX
devnm(VIII): device	name
pwd(I): working directory	name
ncheck(VIII): generate	names from i-numbers
setfil(III): specify Fortran file	name
tty(I): get terminal	name
	nargs(III): argument count
log(III):	natural logarithm
	ncheck(VIII): generate names from i-numbers
	neqn(I): typeset mathematics on terminal
creat(II): create a	new file
newgrp(I): log in to a	new group
fork(II): spawn	new process
next(I):	new standard input for shell procedure
	newgrp(I): log in to a new group
	next(I): new standard input for shell procedure
	nice(I): run a command at low priority
	nice(II): set program priority
	nlist(III): get entries from name list



	nm(I): print name list
	nohup(I): run a command immune to hangups
reset, setexit(III): execute	non-local goto
setjmp, longjmp(III): execute	non-local goto
tmac.name(VII): standard	nroff and troff macro packages
tbl(I): format tables for	nroff or troff
deroff(I): remove	nroff, troff, and eqn constructs
	nroff, troff(I): text formatters
mem, kmem,	null(IV): core memory
rand, srand(III): random	number generator
factor(VI): discover prime factors of a	number
setpgrp(II): set process group	number
size(I): size of an	object file
sky(VI):	obtain ephemerides
od(I):	octal dump
	od(I): octal dump
mm(I): run	off document with PWB/MM
tell(II): get file	offset
	onintr(I): handle interrupts in shell files
man(I): print	on-line documentation
login(I): sign	onto UNIX
dup(II): duplicate an	open file descriptor
fstat(II): get status of	open file
open(II):	open for reading or writing
	open(II): open for reading or writing
plot:	openpl et al.(III): graphics interface
strcpy, strcat, strcmp, strlen(III):	operations on ASCII strings
stty(I): set terminal	options
	othello(VI): a game of dramatic reversals
cpio(I): copy file archives in and	out
ecvt, fcvt(III):	output conversion
locv(III): long	output conversion
a.out(V): assembler and link editor	output
redirect file descriptor 2 (diagnostic	output)...fd2(I):
putc, putw, fcreat, fflush(III): buffered	output
chown(II): change	owner and group of a file
chown(I): change	owner
standard nroff and troff macro	packages...tmac.name(VII):
tail(I): deliver the last	part of a file
	passwd(I): change login password
	passwd(V): password file
crypt(III):	password encoding
passwd(V):	password file
passwd(I): change login	password
patchup(VIII):	patch up a damaged file system
	patchup(VIII): patch up a damaged file system
pexec(III):	path search and execute a file
gmatch(III): match a string with a	pattern (like glob(VIII))
search a file for lines containing a	pattern...egrep(I):
grep(I): search a file for a	pattern
rgrep(I): search a file for a	pattern
	pause(II): indefinite wait
mesg(I):	permit or deny messages

ptx(I): permuted index  
 messages... perror, sys\_errlist, sys\_nerr, errno(III): system  
 udata(II): get per-user data  
 pexec(III): path search and execute a file  
 cat(IV): phototypesetter interface  
 split(I): split a file into pieces  
 tee(I): pipe fitting  
 pipe(II): create an interprocess channel  
 plot: openpl et al.(III): graphics interface  
 plot: t300, t300s, t450(I): graphics filters  
 plot(V): graphics interface  
 fptrap(III): floating point interpreter  
 seek(II): move read/write pointer  
 typo(I): find possible typos  
 pow(III): floating exponentiation  
 bc(I): arbitrary precision interactive language  
 azel(VI): satellite predictions  
 monitor(III): prepare execution profile  
 pr(I): print file  
 factor(VI): discover prime factors of a number  
 date(I): print and set the date  
 cal(I): print calendar  
 sum(I): print checksum of a file  
 pr(I): print file  
 banner(I): print in block letters  
 comm(I): print lines common to two files  
 nm(I): print name list  
 uname(I): print name of current UNIX  
 man(I): print on-line documentation  
 prt(I): print SCCS file  
 cat(I): concatenate and print  
 lp(IV): line printer  
 printf(III): formatted print  
 print  
 printf(III): formatted print  
 vp(I): Versatec print  
 nice(I): run a command at low priority  
 nice(II): set program priority  
 su(I): become privileged user  
 next(I): new standard input for shell procedure  
 70boot(VIII): 11/70 bootstrap procedures  
 unixboot(VIII): UNIX startup and boot procedures  
 init(VIII): process control initialization  
 setgid(II): set process group ID  
 setpgrp(II): set process group number  
 cgetpid(III): return character form of process ID  
 getpid(II): get process identification  
 ps(I): process status  
 times(II): get process times  
 wait(II): wait for process to terminate  
 ptrace(II): process trace  
 setuid(II): set process user ID  
 checklist(V): list of file systems processed by check  
 exit(II): terminate process

fork(II): spawn new	process
shutdown(VIII): terminate all	processing
kill(I): terminate a	process
kill(II): send signal to a	process
m4(I): macro	processor
wait(I): await completion of	process
hmul(III): high-order	product
prof(I): display	prof(I): display profile data
monitor(III): prepare execution	profile data
profil(II): execution time	profile
nice(II): set	profil(II): execution time profile
end, etext, edata(III): last locations in	program priority
make(I): make a	program
lex(I): generate	program
diskboot(VIII): disk bootstrap	programs for simple lexical tasks
tapeboot(VIII): magnetic tape bootstrap	programs
units(I): conversion	programs
	program
	prt(I): print SCCS file
	ps(I): process status
	ptrace(II): process trace
	ptx(I): permuted index
	pump(I): Shell data transfer command
	putc, putw, fcreat, fflush(III): buffered output
	putchar, flush(III): write character
	putc, putw, fcreat, fflush(III): buffered output
mm(I): run off document with	PWB/MM
hasp(VIII):	PWB/UNIX IBM Remote Job Entry
uname(II): get name of current	PWB/UNIX
	pwd(I): working directory name
	qsort(III): quicker sort
qsort(III):	quicker sort
	quiz(I): test your knowledge
	rand, srand(III): random number generator
rand, srand(III):	random number generator
rc(I):	Ratfor compiler
	rc(I): Ratfor compiler
getchar(III):	read character
csw(II):	read console switches
read(II):	read from file
	read(II): read from file
open(II): open for	reading or writing
seek(II): move	read/write pointer
dcat(VIII):	read/write synchronous line
gath(I): gather	real and virtual files
lastcom(VIII): search shell accounting	records
fd2(I):	redirect file descriptor 2 (diagnostic output)
cref(I): make cross	reference listing
reform(I):	reformat text file
	reform(I): reformat text file
expressions...	regcmp, regex(III): compile and execute regular
	regcmp(I): regular expression compile

regen(VIII): regenerate system directories  
 regen(VIII): regenerate system directories  
 regcmp, regex(III): compile and execute regular expressions  
 regcmp(I): regular expression compile  
 regcmp, regex(III): compile and execute regular expressions  
 strip(I): remove symbols and relocation bits  
 hasp(VIII): PWB/UNIX IBM Remote Job Entry  
 rje(IV): DQS-11B interface for remote job entry  
 rmdel(I): remove a delta from an SCCS file  
 rmall(VIII): remove all  
 unlink(II): remove directory entry  
 rmdir(I): remove directory  
 deroff(I): remove nroff, troff, and eqn constructs  
 strip(I): remove symbols and relocation bits  
 rm(I): remove (unlink) files  
 mv(I): move or rename a file  
 uniq(I): report repeated lines in a file  
 df(I): report disk free space  
 uniq(I): report repeated lines in a file  
 reset, setexit(III): execute non-local goto  
 restor(VIII): incremental file system restore  
 restor(VIII): incremental file system restore  
 rsh(I): restricted shell (command interpreter)  
 cgetpid(III): return character form of process ID  
 ttyn(III): return name of current terminal  
 othello(VI): a game of dramatic reversals  
 col(I): filter reverse line feeds  
 rgrep(I): search a file for a pattern  
 send(I): submit RJE job  
 rjstat(I): RJE status and enquiries  
 rje(IV): DQS-11B interface for remote job entry  
 rjstat(I): RJE status and enquiries  
 rmall(VIII): remove all  
 rmdel(I): remove a delta from an SCCS file  
 rmdir(I): remove directory  
 rm(I): remove (unlink) files  
 roff(I): format text  
 romboot(VIII): special ROM bootstrap loaders  
 romboot(VIII): special ROM bootstrap loaders  
 sqrt(III): square root function  
 hp(IV): RP04/RP05/RP06 moving-head disk  
 rp(IV): RP-11/RP03 moving-head disk  
 rp(IV): RP-11/RP03 moving-head disk  
 hs(IV): RS03/RS04 fixed-head disk  
 rsh(I): restricted shell (command interpreter)  
 nice(I): run a command at low priority  
 nohup(I): run a command immune to hangups  
 mm(I): run off document with PWB/MM  
 azel(VI): satellite predictions  
 sa(VIII): Shell accounting  
 break, brk, sbrk(II): change core allocation  
 bfs(I): big file scanner  
 chghist(I): change the history entry of an SCCS delta

delta(I): make an	SCCS delta
comb(I): combine	SCCS deltas
get(I): get generation from	SCCS file
prt(I): print	SCCS file
rmdel(I): remove a delta from an	SCCS file
admin(I): administer	SCCS files
sccsdiff(I): compare two versions of an	SCCS file
sccsfile(V): format of	SCCS file
	sccsdiff(I): compare two versions of an SCCS file
	sccsfile(V): format of SCCS file
alarm(II):	schedule signal after specified time
grep(I):	search a file for a pattern
rgrep(I):	search a file for a pattern
egrep(I):	search a file for lines containing a pattern
fgrep(I):	search a file for lines containing keywords
pexec(III): path	search and execute a file
lastcom(VIII):	search shell accounting records
descend(III):	search UNIX file system directories
	sed(I): stream editor
	seek(II): move read/write pointer
mail(I):	send mail to designated users
kill(II):	send signal to a process
	send(I): submit RJE job
stty(II):	set mode of terminal
setgid(II):	set process group ID
setpgrp(II):	set process group number
setuid(II):	set process user ID
nice(II):	set program priority
tabs(I):	set tabs on terminal
getty(VIII):	set terminal mode
stty(I):	set terminal options
date(I): print and	set the date
stime(II):	set time
setuid(VIII):	set user id of command
ascii(V): map of ASCII character	set
reset,	setexit(III): execute non-local goto
	setfil(III): specify Fortran file name
	setgid(II): set process group ID
	setjmp, longjmp(III): execute non-local goto
	setmnt(VIII): establish mnttab table
	setpgrp(II): set process group number
	setuid(II): set process user ID
	setuid(VIII): set user id of command
	sha(V): Shell accounting file
sha(V):	Shell accounting file
lastcom(VIII): search	shell accounting records
sa(VIII):	Shell accounting
shift(I): adjust	Shell arguments
= (equals) (I):	shell assignment command
rsh(I): restricted	shell (command interpreter)
sh(I):	shell (command interpreter)
pump(I):	Shell data transfer command
onintr(I): handle interrupts in	shell files

while(I): shell iteration command  
 switch(I): shell multi-way branch command  
 next(I): new standard input for shell procedure  
 sh(I): shell (command interpreter)  
 shift(I): adjust Shell arguments  
 shutdown(VIII): terminate all processing  
 login(I): sign onto UNIX  
 alarm(II): schedule signal after specified time  
 kill(II): send signal to a process  
 signal(II): catch or ignore signals  
 signals  
 signal(II): catch or ignore signals  
 lex(I): generate programs for simple lexical tasks  
 sin, cos(III): trigonometric functions  
 size(I): size of an object file  
 size(I): size of an object file  
 sky(VI): obtain ephemerides  
 sleep(I): suspend execution for an interval  
 sleep(III): suspend execution for interval  
 spline(I): interpolate smooth curve  
 sno(I): Snobol interpreter  
 sno(I): Snobol interpreter  
 sort(I): sort or merge files  
 sort(I): sort or merge files  
 qsort(III): quicker sort  
 df(I): report disk free space  
 fork(II): spawn new process  
 mknod(II): make a directory or a special file  
 mknod(VIII): build special file  
 450(I): handle special functions of DASI450 terminal  
 gsi(I): handle special functions of GSI300 terminal  
 hp(I): handle special functions of HP 2640 terminal  
 romboot(VIII): special ROM bootstrap loaders  
 fspec(V): format specification in text files  
 alarm(II): schedule signal after specified time  
 setfil(III): specify Fortran file name  
 spell(I): find spelling errors  
 spell(I): find spelling errors  
 spline(I): interpolate smooth curve  
 split(I): split a file into pieces  
 csplit(I): context split  
 split(I): split a file into pieces  
 sqrt(III): square root function  
 sqrt(III): square root function  
 rand, srand(III): random number generator  
 next(I): new standard input for shell procedure  
 tmac.name(VII): standard nroff and troff macro packages  
 unixboot(VIII): UNIX startup and boot procedures  
 stat(II): get file status  
 ustat(II): get file system statistics  
 rjstat(I): RJE status and enquiries  
 fstat(II): get status of open file  
 gtty(II): get terminal status  
 ps(I): process status

stat(II): get file	status
	stime(II): set time
icheck(VIII): file system	storage consistency check
strcpy,	strcat, strcmp, strlen(III): operations on ASCII strings
strcpy, strcat,	strcmp, strlen(III): operations on ASCII strings
ASCII strings...	strcpy, strcat, strcmp, strlen(III): operations on
sed(I):	stream editor
gmatch(III): match a	string with a pattern (like glob(VIII))
strcmp, strlen(III): operations on ASCII	strings...strcpy, strcat,
strcpy, strcat, strcmp,	strip(I): remove symbols and relocation bits
	strlen(III): operations on ASCII strings
	stty(I): set terminal options
	stty(II): set mode of terminal
send(I):	submit RJE job
	su(I): become privileged user
	sum(I): print checksum of a file
du(I):	summarize disk usage
sync(I): update the	super block
sync(II): update	super-block
sleep(I):	suspend execution for an interval
sleep(III):	suspend execution for interval
csw(II): read console	switches
	switch(I): shell multi-way branch command
strip(I): remove	symbols and relocation bits
dp(IV): DP-11, DU-11	synchronous line interface
dcat(VIII): read/write	synchronous line
	sync(I): update the super block
	sync(II): update super-block
perror,	sys_errlist, sys_nerr, errno(III): system messages
perror, sys_errlist,	sys_nerr, errno(III): system messages
indir(II): indirect	system call
intro(II): introduction to	system calls
check(VIII): file	system consistency check
crash(VIII): what to do when the	system crashes
fsdb(VIII): file	system debugger
descend(III): search UNIX file	system directories
regen(VIII): regenerate	system directories
dcheck(VIII): file	system directory consistency check
dump(VIII): incremental file	system dump
perror, sys_errlist, sys_nerr, errno(III):	system messages
restor(VIII): incremental file	system restore
ustat(II): get file	system statistics
icheck(VIII): file	system storage consistency check
mnttab(V): mounted file	system table
fs(V): format of file	system volume
config(VIII): configure a	system
mkfs(VIII): construct a file	system
mount(II): mount file	system
mount(VIII): mount file	system
patchup(VIII): patch up a damaged file	system
checklist(V): list of file	systems processed by check
umount(II): dismount file	system
umount(VIII): dismount file	system

who(I):	who is on the system
plot:	t300, t300s, t450(I): graphics filters
plot: t300,	t300s, t450(I): graphics filters
plot: t300, t300s,	t450(I): graphics filters
master(V):	master device information table
mnttab(V):	mounted file system table
tbl(I):	format tables for nroff or troff
setmnt(VIII):	establish mnttab table
tabs(I):	set tabs on terminal
	tabs(I): set tabs on terminal
atan, atan2(III):	arc tangent function
tapeboot(VIII):	magnetic tape bootstrap programs
dump(V):	incremental dump tape format
tp(V):	mag tape format
	tapeboot(VIII): magnetic tape bootstrap programs
generate programs for simple lexical	tasks...lex(I):
	tbl(I): format tables for nroff or troff
	tee(I): pipe fitting
greek(V):	graphics for extended TELETYPE Model 37 type-box
HP2640(VII):	Hewlett-Packard 2640 CRT terminal family
ttys(V):	terminal initialization data
tty(IV):	general terminal interface
getty(VIII):	set terminal mode
tty(I):	get terminal name
stty(I):	set terminal options
gtty(II):	get terminal status
450(I):	handle special functions of DASI450 terminal
gsi(I):	handle special functions of GSI300 terminal
hp(I):	handle special functions of HP 2640 terminal
neqn(I):	typeset mathematics on terminal
DASI450, DIABLO 1620, XEROX 1700	terminals...DASI450(VII):
GSI300 (DTC300 or DASI300) hard-copy	terminals...GSI300(VII):
descriptions of commonly-used	terminals...terminals(VII):
TermiNet(VII):	GE TermiNet 300 (and 1200) terminals
TI700(VII):	TI 745, 735, and 725 terminals
stty(II):	set mode of terminal
	terminals(VII): descriptions of commonly-used terminals
tabs(I):	set tabs on terminal
ttyn(III):	return name of current terminal
kill(I):	terminate a process
shutdown(VIII):	terminate all processing
exit(I):	terminate command file
exit(II):	terminate process
wait(II):	wait for process to terminate
TermiNet(VII):	GE TermiNet 300 (and 1200) terminals
	TermiNet(VII): GE TermiNet 300 (and 1200) terminals
quiz(I):	test your knowledge
ed(I):	text editor
reform(I):	reformat text file
fspec(V):	format specification in text files
nroff, troff(I):	text formatters



troff(I):	text formatter
roff(I):	format text
cubic(VI):	three dimensional tic-tac-toe
TI700(VII):	TI 745, 735, and 725 terminals
cubic(VI):	three dimensional tic-tac-toe
TI700(VII):	TI 745, 735, and 725 terminals
tic-tac-toe	
ttt(VI):	the game of tic-tac-toe
time(I):	time a command
profil(II):	execution time profile
localtime, gmtime(III):	convert date and time to ASCII...ctime,
alarm(II):	schedule signal after specified time
time(I):	time a command
time(II):	get date and time
utime(II):	update times in file
times(II):	get process times
stime(II):	set time
times(II):	get process times
time(II):	get date and time
tm(IV):	TM11/TU10 magtape interface
tmac.name(VII):	standard nroff and troff macro packages
tm(IV):	TM11/TU10 magtape interface
tp(I):	manipulate DECtape and magtape
tp(V):	mag tape format
ptrace(II):	process trace
pump(I):	Shell data transfer command
goto(I):	command transfer
tr(I):	transliterate
tr(I):	transliterate
sin, cos(III):	trigonometric functions
deroff(I):	remove nroff, troff, and eqn constructs
tmac.name(VII):	standard nroff and troff macro packages
nroff,	troff(I): text formatter
troff,	troff(I): text formatters
tbi(I):	format tables for nroff or troff
ttt(VI):	the game of tic-tac-toe
tty(I):	get terminal name
tty(IV):	general terminal interface
ttyn(III):	return name of current terminal
ttys(V):	terminal initialization data
ht(IV):	TU16 magtape interface
cmp(I):	compare two files
comm(I):	print lines common to two files
sccsdiff(I):	compare two versions of an SCCS file
graphics for extended TELETYPE Model 37	type-box...greek(V):
file(I):	determine file type
neqn(I):	typeset mathematics on terminal
eqn(I):	typeset mathematics
typo(I):	find possible typos
typo(I):	find possible typos
udata(II):	get per-user data
getpw(III):	get name from UID
umount(II):	dismount file system
umount(VIII):	dismount file system

	uname(I): print name of current UNIX
	uname(II): get name of current PWB/UNIX
	uniq(I): report repeated lines in a file
	units(I): conversion program
descend(III): search	UNIX file system directories
unixboot(VIII):	UNIX startup and boot procedures
	unixboot(VIII): UNIX startup and boot procedures
cu(VIII): call	UNIX
login(I): sign onto	UNIX
uname(I): print name of current	UNIX
rm(I): remove	(unlink) files
	unlink(II): remove directory entry
sync(II):	update super-block
sync(I):	update the super block
utime(II):	update times in file
du(I): summarize disk	usage
setuid(VIII): set	user id of command
getuid(II): get	user identifications
setuid(II): set process	user ID
utmp(V):	user information
wtmp(V):	user login history
mail(I): send mail to designated	users
su(I): become privileged	user
wall(VIII): write to all	users
write(I): write to another	user
	ustat(II): get file system statistics
	utime(II): update times in file
	utmp(V): user information
abs, fabs(III): absolute	value
vp(I):	Versatec print
diffmark(I): mark changes between	versions of a file
sccsdiff(I): compare two	versions of an SCCS file
gath(I): gather real and	virtual files
checking...	volcopy, labelit(VIII): copy filesystems with label
fs(V): format of file system	volume
	vp(I): Versatec print
wait(II):	wait for process to terminate
	wait(I): await completion of process
	wait(II): wait for process to terminate
pause(II): indefinite	wait
	wall(VIII): write to all users
	wc(I): word count
crash(VIII):	what to do when the system crashes
	what(I): identify files
	whatsnew(I): compare file modification dates
	while(I): shell iteration command
who(I):	who is on the system
	who(I): who is on the system
gmatch(III): match a string	with a pattern (like glob(VIII))
volcopy, labelit(VIII): copy filesystems	with label checking
mm(I): run off document	with PWB/MM
wc(I):	word count
pwd(I):	working directory name

cd(I): change working directory  
 chdir(I): change working directory  
 chdir(II): change working directory  
 putchar, flush(III): write character  
     write(II): write on a file  
 wall(VIII): write to all users  
     write(I): write to another user  
     write(I): write to another user  
     write(II): write on a file  
 open(II): open for reading or writing  
     wtmp(V): user login history  
     wump(VI): the game of hunt-the-wumpus  
 DASI450(VII): DASI450, DIABLO 1620, XEROX 1700 terminals  
     xargs(I): construct argument list(s) and execute command  
     yacc(I): yet another compiler-compiler  
     yacc(I): yet another compiler-compiler  
     quiz(I): test your knowledge

**NAME**

450 — handle special functions of DASI450 terminal

**SYNOPSIS**

450

**DESCRIPTION**

450 supports special functions of, and optimizes the use of the DASI450 terminal, or any terminal that is functionally identical, such as the DIABLO 1620 or XEROX 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as *gsi(1)*. 450 can be used to print equations neatly, in the sequence:

```
neqn file ... | nroff | 450
```

NOTE: 450 can be used with the *nroff -s* flag or *.rd* requests when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the RETURN key in these cases, you must use the LINE FEED key to get any response.

However, in most cases, 450 can be eliminated in favor of the following:

```
nroff -T450 files... or nroff -T450-12 files...
```

In a few cases, the additional movement optimization of 450 may produce better-aligned output.

The SPACING switch may be in either 10-pitch or 12-pitch position (but that setting can be overridden dynamically). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

**SEE ALSO**

*graph(1)*, *gsi(1)*, *mesg(1)*, *neqn(1)*, *stty(1)*, *tabs(1)*, *greek(V)*, *DASI450(VII)*, *terminals(VII)*

**BUGS**

Some Greek characters can't be correctly printed in column 1 because the print head cannot be moved to the left from there. If your output contains much Greek and/or reverse line feeds, use friction feed instead of a forms tractor. Although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters, and misaligning the first line after a long set of reverse line feeds.

## NAME

`adb` — debugger

## SYNOPSIS

`adb [-w] [ objfil [ corfil ] ]`

## DESCRIPTION

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

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

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

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

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

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

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

## EXPRESSIONS

- `.` The value of *dot*.
- `+` The value of *dot* incremented by the current increment.
- `^` The value of *dot* decremented by the current increment.
- `"` The last *address* typed.
- integer* An octal number if *integer* begins with a 0; a hexadecimal number if preceded by '#'; otherwise a decimal number.
- integer.fraction* A 32-bit floating point number.
- `'cccc'` The ASCII value of up to 4 characters. `'\'` may be used to escape `'`.
- `< name` The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (q.v.) that are referred to by the letters **a** to **z** or the digits 0 to 9 (see VARIABLES below). If *name* is a register name, then the value of the register is obtained from the system header in *corfil*. The register names are **r0 ... r5 sp pc ps**.
- symbol* A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. `'\'` may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial `'_'` or `'~'` will be prepended to *symbol* if needed.

*routine.name*

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

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

**Monadic operators**

- \* *exp* The contents of the location addressed by *exp* in *corfil*.
- @ *exp* The contents of the location addressed by *exp* in *objfil*.
- *exp* Integer negation.
- ~ *exp* Bitwise complement.

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

- e1 + e2* Integer addition.
- e1 - e2* Integer subtraction.
- e1 \* e2* Integer multiplication.
- e1 % e2* Integer division.
- e1 & e2* Bitwise conjunction.
- e1 | e2* Bitwise disjunction.
- e1 # e2* *e1* rounded up to the next multiple of *e2*.

**COMMANDS**

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

- ? *f* Locations starting at *address* in *objfil* are printed according to the format *f*. *Dot* is incremented by the sum of the increments for each format letter (q.v.).
- / *f* Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for '?'.  
 = *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.)

**Formats**

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 by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

- o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
- O 4 Print 4 bytes in octal.
- q 2 Print in signed octal.
- Q 4 Print long signed octal.
- d 2 Print in decimal.
- D 4 Print long decimal.
- x 2 Print 2 bytes in hexadecimal.

- X 4 Print 4 bytes in hexadecimal.
- u 2 Print as an unsigned decimal number.
- U 4 Print long unsigned decimal.
- f 4 Print the 32-bit value as a floating point number.
- F 8 Print double floating point.
- b 1 Print the addressed byte in octal.
- c 1 Print the addressed character.
- C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
- s n Print the addressed characters until a zero character is reached.
- S n Print a string using the @ escape convention; n is the length of the string including its zero terminator.
- Y 4 Print 4 bytes in date format (see *time(II)*).
- i n Print as PDP-11 instructions; n is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
- a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.
  - / local or global data symbol
  - ? local or global text symbol
  - = local or global absolute symbol
- p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as a.
- t 0 When preceded by an integer, tabs to the next appropriate tab stop. For example, 8t moves to the next 8 space tab stop.
- r 0 Print a space.
- n 0 Print a newline.
- "..." 0 Print the enclosed string.
- ^ *dot* is decremented by the current increment. Nothing is printed.
- + *dot* is incremented by 1. Nothing is printed.
- *dot* is decremented by 1. Nothing is printed.

#### MORE COMMANDS

Here are a few more commands; '[?/]' means the command can start with either '?', for addresses in *objfil*, or '/', for addresses in *corfil*.

#### [?/] 1 *value mask*

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If L is used, then the match is for 4 bytes at a time instead of 2. If no match is found, then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted, then -1 is used.

[?/] **w** *value* ...

*value* is written into the addressed location. If **W** is used then 4 bytes are written, otherwise 2 bytes are written. Odd addresses are not allowed when writing to the sub-process address space.

[?/] **m** *b1 e1 f1* [?/]

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

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

! A shell is called to read the rest of the line following '!'.  
 \$ *modifier*

< *f* Read commands from the file *f* and return.

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

**r** Print the general registers and the instruction addressed by **pc**; *dot* is set to **pc**.

**f** Print the floating registers in single or double length. If the floating point status of **ps** is set to double (0200 bit) then double length is used anyway.

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

**a** ALGOL 68 stack backtrace. If *address* is given then it is taken to be the address of the current frame (instead of **r4**). If *count* is given then only the first *count* frames are printed.

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

**e** The names and values of external variables are printed.

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

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

**o** All integers input are regarded as octal.

**d** Reset integer input as described in EXPRESSIONS.

**q** Exit from *adb*.

**v** Print all non-zero variables in octal.

**m** The values used for mapping addresses into file addresses are printed.

: *modifier*

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

**d** Delete breakpoint at *address*.

**r c** Run *objfil* as a sub-process. If *address* is given explicitly, then the program is entered at this point; otherwise, the program is entered at its standard entry point; *c* specifies how many breakpoints are to be ignored before stopping. Arguments to the sub-process 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 sub-process.

- c s The sub-process is continued with signal *s*. If *address* is given then the sub-process is continued at this address. If no signal is specified then the signal that caused the sub-process to stop is sent. Breakpoint skipping is the same as for *r*.
- s s As for *c* except that the sub-process is single stepped *count* times. If there is no current sub-process then *objfil* is run as a sub-process as for *r*. In this case no signal can be sent; the remainder of the line is treated as arguments to the sub-process.
- k The current sub-process, if any, is terminated.

#### VARIABLES

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

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

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

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

#### ADDRESSES

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

$$b1 \leq \text{address} < e1 \Rightarrow \text{file address} = \text{address} + f1 - b1, \text{ otherwise,}$$

$$b2 \leq \text{address} < e2 \Rightarrow \text{file address} = \text{address} + f2 - b2,$$

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

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

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

#### EXIT STATUS

If the last command was successful then the exit status is zero; otherwise it is non-zero.

#### FILES

/dev/mem  
/dev/swap

**SEE ALSO**

cdb(I), db(I), ptrace(II), a.out(V), core(V)

**BUGS**

- a) A breakpoint set at the entry point is not effective on initial entry to the program.
- b) When single stepping, system calls do not count as an executed instruction.

## NAME

`admin` – administer SCCS files

## SYNOPSIS

`admin` [`-n`] [`-i`[name] [`-rrel`]] [`-t`[name]] [`-fadd-flag`[flag-val]] ... [`-ddelete-flag`] ...  
 [`-aadd-login`] ... [`-eerase-login`] ... [`-h`] [`-z`] name ...

## DESCRIPTION

*Admin* is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments, which begin with “-”, and named files. If a named file doesn’t exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with “s.”), and unreadable files, are silently ignored. If a name of “-” is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument other than *i* and *r* apply independently to each named file.

- `-n` This argument indicates that new files are to be created. This argument must be specified when creating new SCCS files. The *i* argument implies an *n* argument.
- `-i` The name of a file from which the text of an initial delta is to be taken. If this argument is supplied, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this argument is omitted, and the *admin* command creates one or more SCCS files, then their initial deltas must be inserted in the normal manner, using *get* and *delta*(1). Only one SCCS file may be created by an *admin* command on which the *i* argument is supplied.
- `-r` The release into which the initial delta will be inserted. This argument may only be supplied if the *i* argument is also supplied. If this argument is omitted, the initial delta will be inserted into release 1. The level of the initial delta will always be 1.
- `-t` The name of a file from which descriptive text for the SCCS file is to be taken. If this argument is supplied and *admin* is creating a new SCCS file, the descriptive-text filename must also be supplied. In the case of existing SCCS files, if this argument is supplied but the file name is omitted, the descriptive text (if any) currently in the SCCS file will be removed. If the file name is supplied, the text in the file named will replace the descriptive text (if any) currently in the SCCS file.
- `-f` This argument specifies a flag, and, possibly, a value for the flag, to be added to the SCCS file. Several *f* arguments may be supplied on a single *admin* command. The allowable flags and their values are as follows:

- b** The presence of this flag indicates that the use of the **b** argument on a *get* command will cause a branch to be taken in the delta tree.
  - cceil** The "ceiling:" the highest release (less than or equal to 9999) which may be specified by the **r** argument on a *get* with an **e** argument. If this flag is not specified, the ceiling is 9999.
  - dSID** The default SID to be used on a *get* when the **r** argument is not supplied.
  - ffloor** The "floor:" the lowest release (greater than 0) which may be specified by the **r** argument on a *get* with an **e** argument. If this flag is not specified, the floor is 1.
  - i** The presence of this flag causes the "No id keywords (ge6)" message issued by *get* or *delta* to be treated as a fatal error. In the absence of this flag, the message is only a warning.
  - mmod** This flag specifies the module name of the SCCS file. Its value will be used by *get* as the replacement for the **%M%** keyword.
  - ttype** This flag specifies the type of the module. Its value will be used by *get* as a replacement for the **%Y%** keyword.
  - v[pgm]** The presence of this flag indicates that *delta* is to prompt for MR numbers in addition to comments. If the optional value of this flag is present, it specifies the name of an MR number validity checking program.
- d** This argument specifies a flag to be completely removed from an SCCS file. This argument may only be specified when processing existing SCCS files. Several **d** arguments may be supplied on a single *admin* command. See the **f** argument for the allowable flags.
  - a** A login name to be added to the list of logins which may add deltas. Several **a** arguments may be supplied on a single *admin* command. As many logins as desired may be on the list simultaneously. If the list of logins is empty, then anyone may add deltas.
  - e** A login name to be erased from the list of logins. Several **e** arguments may be supplied on a single *admin* command.
  - h** This argument provides a convenient mechanism for checking for corrupted files. With this argument, *admin* will check that the sum of all the characters in the SCCS file (the check-sum) agrees with the sum which is stored in the first line of the file. If the sums are not in agreement a "corrupted file" message will be produced. This argument inhibits writing on the file, so that it will nullify the effect of any other arguments supplied, and is, therefore, only meaningful when processing existing files.
  - z** This argument will cause *admin* to ignore any discrepancy in the check-sum of the SCCS file (see **h** argument), and to replace it with the new one. (The same effect may be had by first editing the SCCS file with *ed(I)* in order to replace the five-character check-sum in the first line of the file with five zeroes. A subsequent invocation of an SCCS command which modifies the file (e.g., *admin*, *delta*), will cause check-sum validation to be by-passed, and a new check-sum to be computed.) The purpose of this is

to correct the check-sum in those files which may have been edited by the user. Note that use of this argument on a truly corrupted file will prevent future detection of the corruption.

#### FILES

The last component of all SCCS file names must be of the form "*s.modulename*". New SCCS files are given mode 444. Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file (see *get(I)*), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file will be deleted, if it exists, and the x-file will be renamed with the name of the SCCS file. This ensures that changes will be made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories will allow only the owner to modify SCCS files contained in the directories. The mode of the SCCS files will prevent any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner, and then the owner may edit the file at will with *ed(I)*.

*Admin* also makes use of the z-file, which is used to prevent simultaneous updates to the SCCS file by different users. See *get(I)* for further information.

#### SEE ALSO

*get(I)*, *delta(I)*, *prt(I)*, *what(I)*, *help(I)*, *ed(I)*, *scsfile(V)*  
*SCCS/PWB User's Manual* by L. E. Bonanni and A. L. Glasser.

#### DIAGNOSTICS

Use *help(I)* for explanations.

**NAME**

**ar** – archive and library maintainer

**SYNOPSIS**

**ar** key [ *posname* ] *afile* name ...

**DESCRIPTION**

*Ar* maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the *ld*(1). It can be used, though, for any similar purpose.

*Key* is one character from the set **drtpmx**, optionally concatenated with **vuabin**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

**d** means delete the named files from the archive file.

**r** means replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If the optional positioning character **a** (also **i** or **b**) is used, then the *posname* argument must be present and specifies a file in the archive after (before for **i** and **b**) which new files are placed. Without **a**, **i**, or **b**, new files are placed at the end.

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

**p** prints the named files in the archive.

**m** moves the named files to the end of the archive. If the options **a**, **i**, or **b** are used, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

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

**v** means verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files.

**n** is accepted with no effect whatsoever.

In all cases, the archive file is created mode 644.

**FILES**

/tmp/v????? temporary  
 /tmp/v1????? temporary  
 /tmp/v2????? temporary

**DIAGNOSTICS**

Most diagnostics are self-explanatory. The message "no space in xxx" means that the file system *xxx* does not have enough space to contain the temporary files or the new archive file.

**SEE ALSO**

*ld*(1), *archive*(V)

**BUGS**

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

**NAME**

as — assembler

**SYNOPSIS**

as [ - ] [ -o objfil ] name...

**DESCRIPTION**

*As* assembles the concatenation of the named files. If the optional first argument `-` is used, all undefined symbols in the assembly are treated as global.

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

**FILES**

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

**SEE ALSO**

ld(I), nm(I), db(I), a.out(V), *UNIX Assembler Reference Manual* by D. M. Ritchie.

**DIAGNOSTICS**

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

)	Parentheses error
]	Parentheses error
<	String not terminated properly
*	Indirection used illegally
.	Illegal assignment to '.'
A	Error in address
B	Branch instruction is odd or too remote
E	Error in expression
F	Error in local ('f' or 'b') type symbol
G	Garbage (unknown) character
I	End of file inside an if
M	Multiply defined symbol as label
O	Word quantity assembled at odd address
P	'.' different in pass 1 and 2
R	Relocation error
U	Undefined symbol
X	Syntax error

**BUGS**

Symbol table overflow is not checked. `x` errors can cause incorrect line numbers in following diagnostics.

**NAME**

**banner** — print in block letters

**SYNOPSIS**

**banner** arg ...

**DESCRIPTION**

*Banner* writes characters as large block letters, 7 characters by 7 characters, on the standard output file. Each argument may be up to ten characters, and is printed on a separate row.



**NAME**

bas — basic

**SYNOPSIS**

bas [ file ]

**DESCRIPTION**

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

statement  
integer statement

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

Statements have the following syntax:

**expression**

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

**comment ...**

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

**done**

Return to system level.

**draw** expression expression expression

A line is drawn on the Tektronix 611 display '/dev/vt0' from the current display position to the XY co-ordinates specified by the first two expressions. The scale is zero to one in both X and Y directions. If the third expression is zero, the line is invisible. The current display position is set to the end point.

**display** list

The list of expressions and strings is concatenated and displayed (i.e. printed) on the 611 starting at the current display position. The current display position is not changed.

**dump**

The name and current value of every variable is printed.

**edit**

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

**erase**

The 611 screen is erased.

**for** name = expression expression statement

**for** name = expression expression

...

**next**

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

second expression.

**goto** expression

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

**if** expression statement

**if** expression

...

[ **else**

... ]

**fi**

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

**list** [expression [expression]]

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

**print** list

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

**prompt** list

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

**return** [expression]

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

**run**

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

**save** [expression [expression]]

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

Expressions have the following syntax:

**name**

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

**number**

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

( expression )

Parentheses are used to alter normal order of evaluation.

**\_** expression

The result is the negation of the expression.

expression operator expression

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

expression ( [expression [ , expression] ... ] )

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

name [ expression [ , expression ] ... ]

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

The following is the list of operators:

=

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

& |

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

< <= > >= == <>

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

+ -

Add and subtract.

\* /

Multiply and divide.

^

Exponentiation.

The following is a list of builtin functions:

**arg(i)**

is the value of the *i*-th actual parameter on the current level of function call.

**exp(x)**

is the exponential function of *x*.

**log(x)**

is the natural logarithm of *x*.

**sqr(x)**

is the square root of *x*.

**sin(x)**

is the sine of *x* (radians).

**cos(x)**

is the cosine of  $x$  (radians).

**atn(x)**

is the arctangent of  $x$ . Its value is between  $-\pi/2$  and  $\pi/2$ .

**rnd()**

is a uniformly distributed random number between zero and one.

**expr()**

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

**abs(x)**

is the absolute value of  $x$ .

**int(x)**

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

**FILES**

/tmp/btm?	temporary
b.out	save file

**DIAGNOSTICS**

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

**BUGS**

Has been known to give core images.

## NAME

bc — arbitrary precision interactive language

## SYNOPSIS

bc [ -l ] [ file ... ]

## DESCRIPTION

*Bc* is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The '-l' argument stands for the name of a library of mathematical subroutines which contains sine (named 's'), cosine ('c'), arctangent ('a'), natural logarithm ('l'), and exponential ('e'). The syntax for *bc* programs is as follows; E means expression, S means statement.

## Comments

are enclosed in /\* and \*/.

## Names

letters a-z

array elements: letter[E]

The words 'ibase', 'obase', and 'scale'

## Other operands

arbitrarily long numbers with optional sign and decimal point.

( E )

sqrt ( E )

<letter> ( E , ... , E )

## Operators

+ - \* / % ^

++ -- (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 are exemplified by

```
define <letter> ( <letter> ,..., <letter> ) {
    auto <letter>, ... , <letter>
    S; ... S
    return ( E )
}
```

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations. Assignments to *ibase* or *obase* set

the input and output number radix respectively.

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

For example

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

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

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

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

#### FILES

/usr/lib/lib.b mathematical library

#### SEE ALSO

dc(1)

*C Reference Manual* by D. M. Ritchie.

*BC - An Arbitrary Precision Desk Calculator Language* by L. L. Cherry and R. Morris.

#### BUGS

No &&, || yet.

*for* statement must have all three E's

*quit* is interpreted when read, not when executed.

## NAME

`bdiff` - big diff

## SYNOPSIS

`bdiff` name1 name2

## DESCRIPTION

*Bdiff* is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*(1). *Bdiff* splits the files into 4000-line segments, and invokes *diff*(1) on corresponding segments (4000-lines is a reasonable upper limit for *diff*(1)). If *name1* (*name2*) is "-", the standard input is read. The output of *bdiff* is exactly that of *diff*(1), with line numbers adjusted to account for the segmenting (that is, to make it look as if the files had been processed whole).

Note that unlike *diff*(1), *bdiff* supports no optional keyletter arguments. In addition, because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

## FILES

/tmp/bd????

## SEE ALSO

*diff*(1)

## DIAGNOSTICS

Use *help*(1) for explanations.

## NAME

bfs — big file scanner

## SYNOPSIS

**bfs** [ - ] name

## DESCRIPTION

*Bfs* is (almost) like *ed*(I) except that it is read-only and processes much bigger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 255 characters per line. *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer.

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 is typed as in *ed*. 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* 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. Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regex*(III)). 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*, *n*, *p*, *q*, *w*, *=*, *!* and null commands operate as described under *ed*. 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*, *xi* 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*. *Xf* commands may be nested to a depth of 10.

*xo* [file]

Further output from the *p* and null commands is diverted to the named file, which, if necessary, is created mode 666. Plain *xo* diverts output back 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 newline, 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 the named 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 doesn't 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 the label. This command is the only one that doesn't 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: 0-9][optional spaces][value]

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

Another feature of the *xv* command is that the first line of output from a UNIX 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 unix command (!UNIX command) and branch on a zero 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
xbr 1

xv45
: 1
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbr 1
```

**xc** [*switch*]

If *switch* is 1, output from the *p* and null commands is crunched; if *switch* is 0 it isn't. Plain 'xc' reverses the switch. Initially the switch is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

**SEE ALSO**

ed(I), regex(III)

**DIAGNOSTICS**

'?' for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

**NAME**

cal – print calendar

**SYNOPSIS**

cal [ month ] year

**DESCRIPTION**

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

Try September 1752.

**BUGS**

The year is always considered to start in January even though this is historically naive.

**NAME**

**cat** - concatenate and print

**SYNOPSIS**

**cat** [ **-s** ] [ **-u** ] file ...

**DESCRIPTION**

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

**cat file**

prints the file, and

**cat file1 file2 >file3**

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

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

The **-s** flag suppresses the error messages that *cat* would otherwise give for non-existent (or unreadable) files. The **-u** flag causes *cat* to work in an unbuffered fashion (read one character, then write that character).

**SEE ALSO**

**pr(I)**, **cp(I)**

**DIAGNOSTICS**

file not found

**BUGS**

**cat x y >x** and **cat x y >y** cause strange results (because of *sh(I)*).

**NAME**

*cb* - C beautifier

**SYNOPSIS**

***cb***

**DESCRIPTION**

*cb* reads a C program from the standard input, adds the proper indentation, and writes it on the standard output.

## NAME

`cc` - C compiler

## SYNOPSIS

`cc [-c] [-p] [-f] [-Dn=v] [-Idir] [-O] [-S] [-P] [-Un] files ...`

## DESCRIPTION

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

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

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

- `-c` Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.
- `-p` Arrange for the compiler to produce code which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls the `monitor(III)` subroutine at the start and arranges to write out a `mon.out` file at normal termination of execution of the object program. An execution profile can then be generated by use of `prof(1)`.
- `-f` In systems without hardware floating-point, use a version of the C compiler which handles floating-point constants and loads the object program with the floating-point interpreter. Do not use if the hardware is present.
- `-D` The name `n` is *defined*, and is given the value `v`, if specified.
- `-O` Invoke an object-code optimizer.
- `-S` Compile the named C programs, and leave the assembler-language output on corresponding files suffixed `.s`.
- `-P` Run only the macro preprocessor on the named C programs, and leave the output on corresponding files suffixed `.i`.
- `-U` The name `n` is *undefined*.
- `-I` The `include` preprocessor statement looks in directory `dir` if it can't find the specified file in the local directory or in `/usr/include`.

Other arguments are taken to be either loader flag arguments, or C-compatible object programs, typically produced by an earlier `cc` run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name `a.out`. If desired, a different name can be used; see the `-o` option of `ld(1)`.

## FILES

<code>file.c</code>	input file
<code>file.o</code>	object file
<code>a.out</code>	loaded output
<code>/tmp/ctm?</code>	temporary
<code>/lib/c[01]</code>	compiler
<code>/lib/fc[01]</code>	floating-point compiler

/lib/c2	optional optimizer
/lib/cpp	pre-processor
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	runtime startoff of profiling
/lib/fcrt0.o	runtime startoff for floating-point interpretation
/lib/libc.a	C library; see section III.
/lib/liba.a	Assembler library used by some routines in libc.a

**SEE ALSO**

*C Reference Manual* by D. M. Ritchie.  
*Programming in C - A Tutorial* by B. W. Kernighan.  
adb(I), cdb(I), ld(I), prof(I), monitor(III)

**DIAGNOSTICS**

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

**NAME**

**cd** – change working directory

**SYNOPSIS**

**cd** directory

**DESCRIPTION**

*Cd* is an alias for *chdir*(1).

**SEE ALSO**

*chdir*(1), *sh*(1), *pwd*(1)



## NAME

`cdb` — C debugger

## SYNOPSIS

`cdb [ a.out [ core ] ]`

## DESCRIPTION

`Cdb` is a debugger for use with C programs. It is useful for both post-mortem and interactive debugging. An important feature of `cdb` is that even in the interactive case no advance planning is necessary to use it; in particular it is not necessary to compile or load the program in any special way nor to include any special routines in the object file.

The first argument to `cdb` is an object program, preferably containing a symbol table; if not given "a.out" is used. The second argument is the name of a core-image file; if it is not given, "core" is used. The core file need not be present.

Commands to `cdb` consist of an address, followed by a single command character, possibly followed by a command modifier. Usually if no address is given the last-printed address is used. An address may be followed by a comma and a number, in which case the command applies to the appropriate number of successive addresses.

Addresses are expressions composed of names, decimal numbers, and octal numbers (which begin with "0"), separated by "+" and "-". Evaluation proceeds left-to-right.

Names of external variables are written just as they are in C. For various reasons the external names generated by C all begin with an underscore, which is automatically tacked on by `cdb`. Currently it is not possible to suppress this feature, so symbols (defined in assembly-language programs) which do not begin with underscore are inaccessible.

Variables local to a function (automatic, static, and arguments) are accessible by writing the name of the function, a colon ":", and the name of the local variable (e.g. "main:argc"). There is no notion of the "current" function; its name must always be written explicitly.

A number which begins with "0" is taken to be octal; otherwise numbers are decimal, just as in C. There is no provision for input of floating numbers.

The construction "name[expression]" assumes that *name* is a pointer to an integer and is equivalent to the contents of the named cell plus twice the expression. Notice that *name* has to be a genuine pointer and that arrays are not accessible in this way. This is a consequence of the fact that types of variables are not currently saved in the symbol table.

The command characters are:

`/m` print the addressed words. *m* indicates the mode of printout; specifying a mode sets the mode until it is explicitly changed again:

- `o` octal (default)
- `i` decimal
- `f` single-precision floating-point
- `d` double-precision floating-point

`\` Print the specified bytes in octal.

`=` print the value of the addressed expression in octal.

`'` print the addressed bytes as characters. Control and non-ASCII characters are escaped in octal.

- " take the contents of the address as a pointer to a sequence of characters, and print the characters up to a null byte. Control and non-ASCII characters are escaped as octal.
- & Try to interpret the contents of the address as a pointer, and print symbolically where the pointer points. The printout contains the name of an external symbol and, if required, the smallest possible positive offset. Only external symbols are considered.
- ? Interpret the addressed location as a PDP-11 instruction.
- \$*m* If no *m* is given, print a stack trace of the terminated or stopped program. The last call made is listed first; the actual arguments to each routine are given in octal. (If this is inappropriate, the arguments may be examined by name in the desired format using "/".) If *m* is "r", the contents of the PDP-11 general registers are listed. If *m* is "f", the contents of the floating-point registers are listed. In all cases, the reason why the program stopped or terminated is indicated.
- %*m* According to *m*, set or delete a breakpoint, or run or continue the program:

**b** An address within the program must be given; a breakpoint is set there. Ordinarily, breakpoints will be set on the entry points of functions, but any location is possible as long as it is the first word of an instruction. (Labels don't appear in the symbol table yet.) Stopping at the actual first instruction of a function is undesirable because to make symbolic printouts work, the function's save sequence has to be completed; therefore *cdb* automatically moves breakpoints at the start of functions down to the first real code.

It is impossible to set breakpoints on pure-procedure programs ( `-n` flag on *cc* or *ld* (I)) because the program text is write-protected.

- d** An address must be given; the breakpoint at that address is removed.
- r** Run the program being debugged. Following the "%r", arguments may be given; they cannot specify I/O redirection (">", "<") or filters. No address is permissible, and the program is restarted from scratch, not continued. Breakpoints should have been set if any were desired. The program will stop if any signal is generated, such as illegal instruction (including simulated floating point), bus error, or interrupt (see *signal*(II)); it will also stop when a breakpoint occurs and in any case announce the reason. Then a stack trace can be printed, named locations examined, etc.
- c** Continue after a breakpoint. It is possible but probably useless to continue after an error since there is no way to repair the cause of the error.

#### SEE ALSO

*cc*(I), *db*(I), *C Reference Manual* by D. M. Ritchie.

#### BUGS

Use caution in believing values of register variables at the lowest levels of the call stack; the value of a register is found by looking at the place where it was supposed to have been saved by the callee.

Some things are still needed to make *cdb* uniformly better than *db*: non-C symbols, patching files, patching core images of programs being run. It would be desirable to have the types of variables around to make the correct style printout more automatic. Structure members should be available.

Naturally, there are all sorts of neat features not handled, like conditional breakpoints, optional stopping on certain signals (like illegal instructions, to allow breakpointing of simulated floating-point programs).

**NAME**

`chdir` — change working directory

**SYNOPSIS**

`chdir` *directory*  
`cd` *directory*

**DESCRIPTION**

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

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

*Cd* is a synonym for *chdir* and acts identically.

**SEE ALSO**

`sh(1)`, `pwd(1)`

**NAME**

chghist — change the history entry of an SCCS delta

**SYNOPSIS**

**chghist** —rSID name ...

**DESCRIPTION**

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

If a directory is named, *chghist* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with "s."), and unreadable files, are silently ignored. If a name of "-" is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files, and unreadable files, are silently ignored.

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

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

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

**FILES**

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

**SEE ALSO**

*admin*(I), *get*(I), *delta*(I), *prt*(I), *help*(I), *scsfile*(V)  
*SCCS/PWB User's Manual* by L. E. Bonanni and A. L. Glasser.

**DIAGNOSTICS**

Use *help*(I) for explanations.

**NAME**

chgrp — change group

**SYNOPSIS**

**chgrp** group file ...

**DESCRIPTION**

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

**SEE ALSO**

chown(1), group(V)

**FILES**

/etc/group

**NAME**

chmod – change mode

**SYNOPSIS**

**chmod** octal file ...

**DESCRIPTION**

The octal mode replaces the mode of each of the files. The mode is constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit for shared, pure-procedure programs (see below)
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

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

If an executable file is set up for sharing (“-n’ of *ld(I)*), then mode 1000 prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time. Ability to set this bit is restricted to the super-user since swap space is consumed by the images; it is only worth while for heavily used commands.

**SEE ALSO**

ls(I), chmod(II)

**NAME**

chown – change owner

**SYNOPSIS**

**chown** owner file ...

**DESCRIPTION**

The user-ID of the files is changed to *owner*. The owner may be either a decimal UID or a login name found in the password file.

**FILES**

/etc/passwd

**SEE ALSO**

chgrp(1), passwd(V)

**NAME**

`cmp` – compare two files

**SYNOPSIS**

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

**DESCRIPTION**

The two files are compared. (If *file1* is '-', the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted. Moreover, return code 0 is yielded for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

**Options:**

- `-l` Print the byte number (decimal) and the differing bytes (octal) for each difference.
- `-s` Print nothing for differing files; return codes only.

**SEE ALSO**

`diff(1)`, `comm(1)`



**NAME**

`col` – filter reverse line feeds

**SYNOPSIS**

`col`

**DESCRIPTION**

*Col* reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-7). *Col* is particularly useful for filtering multicolumn output made with the `.rt` command of *nroff*.

**SEE ALSO**

`nroff(1)`

**BUGS**

Can't back up more than 102 lines.

The input file must not have ASCII tab characters; *col* does not handle them properly (see *reform(1)*).

## NAME

comb - combine SCCS deltas

## SYNOPSIS

comb [-o] [-s] [-psid] [-clist] name ...

## DESCRIPTION

*Comb* generates a shell procedure (see *sh*(I)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with "s."), and unreadable files are silently ignored. If a name of "-" is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files, and unreadable files are silently ignored.

The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

- p The SCCS identification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.
- c A list (see *get*(I) for the syntax of a list) of deltas to be preserved. All other deltas are discarded.
- o This argument causes the reconstructed file to be accessed at the release of the delta to be created for each "get -e" generated. Without this argument, the reconstructed file is accessed at the most recent ancestor for each "get -e" generated. Use of the o keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.
- s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file, the file name, size after combining, original size, and percentage change computed by:
 
$$100 * (\text{original} - \text{combined}) / \text{original}$$
 (Sizes are in blocks.) We recommend that before any SCCS files are actually combined one should use this option to determine exactly how much space is saved by the combining process.

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

## FILES

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

## SEE ALSO

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

**DIAGNOSTICS**

Use *help*(1) for explanations.

**BUGS**

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

**NAME**

**comm** - print lines common to two files

**SYNOPSIS**

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

**DESCRIPTION**

*Comm* reads *file1* and *file2*, which should be sorted in the same order, and produces a three column output: lines only in *file1*; lines only in *file2*; and lines in both files. The filename '-' means the standard input.

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

**SEE ALSO**

**cmp(I)**, **diff(I)**, **uniq(I)**

**NAME**

cp — copy

**SYNOPSIS**

cp file1 file2

**DESCRIPTION**

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

If *file2* is a directory, then the target file is a file in that directory with the file-name of *file1*.

It is forbidden to copy a file onto itself.

**SEE ALSO**

cpx(I), ln(I), cat(I), pr(I), mv(I)

## NAME

`cpio` — copy file archives in and out

## SYNOPSIS

```
cpio -o[v]
cpio -i[drtuv] [ pattern ]
cpio -p[dlruv] [ pattern ] directory
```

## DESCRIPTION

`Cpio -o` (copy out) reads the standard input for 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 the product of a previous "`cpio -o`", files whose names are selected by a *pattern* given in the name-generating syntax of *sh*(I). The *pattern* meta-characters '?', '\*', '[...]' will match '/' characters. The *pattern* argument defaults to "\*".

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

The options are:

- d** Directories are to be created as needed.
- r** Interactively *rename* files. If the user types a null line, the file is skipped.
- t** Print a *table of contents* of the input. No files are created.
- u** Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).
- v** *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like an "`ls -l`" (see *ls*(I)).
- l** Whenever possible, link files rather than copying them. Usable only with the **-p** option.
- m** Retain previous file modified time (only for the super-user).

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

```
ls | cpio -o >/dev/mt0
chdir olddir
find . -print | cpio -pdl newdir
```

## SEE ALSO

*ar*(I), *cpio*(V)

## BUGS

Path names are restricted to 128 characters.

If there are too many unique linked files, the program runs out of memory to keep track of them and subsequent linking information is lost.

**NAME**

`cpx` — copy a file exactly

**SYNOPSIS**

`cpx` - [*file1* | -] [*file2* | -]

**DESCRIPTION**

*Cpx* copies *file 1* onto *file 2*. The mode, owner and time of last modification of the source file are preserved.

Either *file1* or *file2* may be represented as a “-”, which uses the standard UNIX input/output pipe mechanism, instead of the corresponding file. A file read from a pipe or written to a pipe will be preceded with a header, containing the mode, owner, time of last modification, number of characters, and a summed total of the characters in the file. The case where a pipe is read and a file is written, both the number of characters and the summed total are compared to similar values after the copy. If there are no differences between the comparisons, the message “ok” is printed.

*Cpx* prohibits copying a file onto itself.

*Cpx* does not allow *file1* to be a directory. If *file2* is a directory, then the target file is a file in that directory with the file name of *file1*.

Examples to copy a file to the current directory:

```
cpx ../file1 - | cpx - .
cpx ../file1 .
cpx ../file1 file2
```

**SEE ALSO**

`cp(1)`

## NAME

`cref` - make cross reference listing

## SYNOPSIS

`cref [ -acilnostux123 ] name ...`

## DESCRIPTION

*Cref* makes a cross reference listing of program files in assembler or C format. The files named as arguments in the command line are searched for symbols in the appropriate syntax.

The output report is in four columns:

(1)	(2)	(3)	(4)
symbol	file	see	text as it appears in file
		below	

*Cref* uses either an *ignore* file or an *only* file. If the `-i` option is given, the next argument is taken to be an *ignore* file; if the `-o` option is given, the next argument is taken to be an *only* file. *Ignore* and *only* files are lists of symbols separated by new lines. All symbols in an *ignore* file are ignored in columns (1) and (3) of the output. If an *only* file is given, only symbols in that file appear in column (1). At most one of `-i` and `-o` may be used. The default setting is `-i`. Assembler predefined symbols or C keywords are ignored.

The `-s` option causes current symbols to be put in column 3. In the assembler, the current symbol is the most recent name symbol; in C, the current function name. The `-l` option causes the line number within the file to be put in column 3.

The `-t` option causes the next available argument to be used as the name of the intermediate temporary file (instead of `/tmp/crt??`). The file is created and is not removed at the end of the process.

Options:

- a** assembler format (default)
- c** C format input
- i** use *ignore* file (see above)
- l** put line number in col. 3 (instead of current symbol)
- n** omit column 4 ("no context")
- o** use *only* file (see above)
- s** current symbol in col. 3 (default)
- t** user supplied temporary file
- u** print only symbols that occur exactly once
- x** print only C external symbols
- 1** sort output on column 1 (default)
- 2** sort output on column 2
- 3** sort output on column 3

## FILES

<code>/tmp/crt??</code>	temporaries
<code>/usr/lib/aign</code>	default assembler <i>ignore</i> file
<code>/usr/lib/atab</code>	grammar table for assembler files
<code>/usr/lib/cign</code>	default C <i>ignore</i> file
<code>/usr/bin/crpost</code>	post processor
<code>/usr/lib/ctab</code>	grammar table for C files



/usr/bin/upost post processor for **-u** option  
/bin/sort used to sort temporaries

**SEE ALSO**

as(I), cc(I)

**NAME**

`crypt` — encode/decode

**SYNOPSIS**

`crypt` [ password ]

**DESCRIPTION**

*crypt* simulates a cryptographic machine.

*crypt* reads from the standard input file and writes on the standard output. It is thus suitable for use as a filter. For a given password, the encryption process is idempotent; that is,

```
crypt znorkle <clear >cypher  
crypt znorkle <cypher
```

will print the clear.

## NAME

`csplit` - context split

## SYNOPSIS

`csplit` [ `-s` ] [ `-f prefix` ] file [RE01 RE02 ... REn]

## DESCRIPTION

*Csplit* reads *file* and separates it into  $n+1$  sections, defined by the regular expressions RE01, ... , REn, where  $n$  is less than 100. If the `-f` option is used, the sections are placed in *prefix*00 ... *prefix* $n$ . The default is *xx*00 ... *xx* $n$ . These sections get the following pieces of *file*:

- 00: from the start of the file up to (but not including) the first line matched by RE01
- 01: from the line matched by RE01 up to the first line that is matched by RE02
- ⋮
- $n+1$ : line matched by REn to the end of the file

Enclose by double quotes (") all RE's that contain blanks or other characters meaningful to the Shell.

*Csplit* tells the size of the original file, as well as of each "split" file as it creates it. It also prints any appropriate diagnostics. If the `-s` option is present, *csplit* suppresses the printing of all character counts.

## EXAMPLE:

```
csplit -f zz file "procedure division" par5. par16.
```

After editing the "split" files, they can be recombined as follows:

```
cat zz0[0-3] >file
```

It should be noted that *csplit* does not affect in any way the original file. The responsibility for removing it is the user's.

## SEE ALSO

`ed(1)`, `sh(1)`

## NAME

date – print and set the date

## SYNOPSIS

date [ mmddhhmm[yy] ] [ +format ]

## DESCRIPTION

If no argument is given, or if the argument begins with “+”, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

**date 10080045**

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

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(III)*. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by “%” and will be replaced in the output by its corresponding value. A single “%” is encoded by “%%”. All other characters are copied to the output without change. The string is always terminated with a newline character.

## Field Descriptors:

**n** insert a newline character  
**t** insert a tab character  
**m** month of year – 01 to 12  
**d** day of month – 01 to 31  
**y** last 2 digits of year – 00 to 99  
**H** hour – 00 to 23  
**M** minute – 00 to 59  
**S** second – 00 to 59  
**j** julian date – 001 to 366  
**w** day of week – Sunday = 0  
**a** abbreviated weekday – Sun to Sat  
**h** abbreviated month – Jan to Dec  
**r** time in AM / PM notation

For example:

**date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"**

would generate as output:

**DATE: 08/01/76**  
**TIME: 14:45:05**

## DIAGNOSTICS

“No permission” if you aren’t the super-user and you try to change the date; “bad conversion” if the date set is syntactically incorrect; “invalid option” if the field descriptor is not recognizable.

## FILES

/dev/kmem

## NAME

db - debug

## SYNOPSIS

db [ core [ namelist ] ] [ - ]

## DESCRIPTION

Unlike many debugging packages (including the Digital Equipment Corporation's ODT, on which *db* is loosely based), *db* is not loaded as part of the core image which it is used to examine; instead it examines files. Typically, the file will be either a core image produced after a fault or the binary output of the assembler. *Core* is the file being debugged; if omitted *core* is assumed. *Namelist* is a file containing a symbol table. If it is omitted, the symbol table is obtained from the file being debugged, or if not there from *a.out*. If no appropriate name list file can be found, *db* can still be used but some of its symbolic facilities become unavailable.

For the meaning of the optional third argument, see the last paragraph below.

The format for most *db* requests is an address followed by a one character command. Addresses are expressions built up as follows:

1. A name has the value assigned to it when the input file was assembled. It may be relocatable or not depending on the use of the name during the assembly.
2. An octal number is an absolute quantity with the appropriate value.
3. A decimal number immediately followed by '.' is an absolute quantity with the appropriate value.
4. An octal number immediately followed by *r* is a relocatable quantity with the appropriate value.
5. The symbol . indicates the current pointer of *db*. The current pointer is set by many *db* requests.
6. A \* before an expression forms an expression whose value is the number in the word addressed by the first expression. A \* alone is equivalent to '\*.'
7. Expressions separated by + or blank are expressions with value equal to the sum of the components. At most one of the components may be relocatable.
8. Expressions separated by - form an expression with value equal to the difference to the components. If the right component is relocatable, the left component must be relocatable.
9. Expressions are evaluated left to right.

Names for registers are built in:

r0 ... r5 sp pc fr0 ... fr5

These may be examined. Their values are deduced from the contents of the stack in a core image file. They are meaningless in a file that is not a core image.

If no address is given for a command, the current address (also specified by ".") is assumed. In general, "." points to the last word or byte printed by *db*.

There are *db* commands for examining locations interpreted as numbers, machine instructions, ASCII characters, and addresses. For numbers and characters, either bytes or words may be examined. The following commands are used to examine the specified file.

- / The addressed word is printed in octal.
- \ The addressed byte is printed in octal.
- " The addressed word is printed as two ASCII characters.
- ' The addressed byte is printed as an ASCII character.
- ` The addressed word is printed in decimal.
- ? The addressed word is interpreted as a machine instruction and a symbolic form of the instruction, including symbolic addresses, is printed. Often, the result will appear exactly as it was written in the source program.
- & The addressed word is interpreted as a symbolic address and is printed as the name of the symbol whose value is closest to the addressed word, possibly followed by a signed offset.
- <nl> (i. e., the character "new line") This command advances the current location counter "." and prints the resulting location in the mode last specified by one of the above requests.
- ^ This character decrements "." and prints the resulting location in the mode last selected one of the above requests. It is a converse to <nl>.
- % Exit.

Odd addresses to word-oriented commands are rounded down. The incrementing and decrementing of "." done by the <nl> and ^ requests is by one or two depending on whether the last command was word or byte oriented.

The address portion of any of the above commands may be followed by a comma and then by an expression. In this case that number of sequential words or bytes specified by the expression is printed. "." is advanced so that it points at the last thing printed.

There are two commands to interpret the value of expressions.

- = When preceded by an expression, the value of the expression is typed in octal. When not preceded by an expression, the value of "." is indicated. This command does not change the value of ".".
- : An attempt is made to print the given expression as a symbolic address. If the expression is relocatable, that symbol is found whose value is nearest that of the expression, and the symbol is typed, followed by a sign and the appropriate offset. If the value of the expression is absolute, a symbol with exactly the indicated value is sought and printed if found; if no matching symbol is discovered, the octal value of the expression is given.

The following command may be used to patch the file being debugged.

- ! This command must be preceded by an expression. The value of the expression is stored at the location addressed by the current value of ".". The opcodes do not appear in the symbol table, so the user must assemble them by hand.

The following command is used after a fault has caused a core image file to be produced.

- \$ causes the fault type and the contents of the general registers and several other registers to be printed both in octal and symbolic format. The values are as they were at the time of the fault.

For some purposes, it is important to know how addresses typed by the user correspond with locations in the file being debugged. The mapping algorithm employed by *db* is non-trivial for two reasons: First, in an *a.out* file, there is a 20(8) byte header which will not appear when the file is loaded into core for execution. Therefore, apparent location 0 should correspond with actual file offset 20. Second, addresses in core images do not correspond with the addresses used by the program because in a core

image there is a header containing the system's per-process data for the dumped process, and also because the stack is stored contiguously with the text and data part of the core image rather than at the highest possible locations. *Db* obeys the following rules:

If exactly one argument is given, and if it appears to be an **a.out** file, the 20-byte header is skipped during addressing, i.e., 20 is added to all addresses typed. As a consequence, the header can be examined beginning at location `-20`.

If exactly one argument is given and if the file does not appear to be an **a.out** file, no mapping is done.

If zero or two arguments are given, the mapping appropriate to a core image file is employed. This means that locations above the program break and below the stack effectively do not exist (and are not, in fact, recorded in the core file). Locations above the user's stack pointer are mapped, in looking at the core file, to the place where they are really stored. The per-process data kept by the system, which is stored in the first part of the core file, cannot currently be examined (except by **\$**).

If one wants to examine a file which has an associated name list, but is not a core image file, the last argument `"-"` can be used (actually the only purpose of the last argument is to make the number of arguments not equal to two). This feature is used most frequently in examining the memory file `/dev/mem`.

#### SEE ALSO

`as(I)`, `core(V)`, `a.out(V)`, `od(I)`

#### DIAGNOSTICS

"File not found" if the first argument cannot be read; otherwise `"?"`.

#### BUGS

There should be some way to examine the registers and other per-process data in a core image; also there should be some way of specifying double-precision addresses. It does not know yet about shared text segments.

**NAME**

`dc` — desk calculator

**SYNOPSIS**

`dc [ file ]`

**DESCRIPTION**

`Dc` is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of `dc` is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

**number**

The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore `_` to input a negative number. Numbers may contain decimal points.

`+ - * % ^`

The top two values on the stack are added (`+`), subtracted (`-`), multiplied (`*`), divided (`/`), remaindered (`%`), or exponentiated (`^`). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

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

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

`d` The top value on the stack is duplicated.

`p` The top value on the stack is printed. The top value remains unchanged.

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

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

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

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

`<x >x =x`

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

`v` replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

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

`c` All values on the stack are popped.

`i` The top value on the stack is popped and used as the number radix for further input.



- o** The top value on the stack is popped and used as the number radix for further output.
- 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.
- ?** A line of input is taken from the input source (usually the console) and executed.

An example which prints the first ten values of  $n!$  is:

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

#### SEE ALSO

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

#### DIAGNOSTICS

- (x) ? for unrecognized character x.
- (x) ? for not enough elements on the stack to do what was asked by command x.
- 'Out of space' when the free list is exhausted (too many digits).
- 'Out of headers' for too many numbers being kept around.
- 'Out of pushdown' for too many items on the stack.
- 'Nesting Depth' for too many levels of nested execution.

## NAME

dd – convert and copy a file

## SYNOPSIS

dd [option=value] ...

## DESCRIPTION

*Dd* copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

<i>option</i>	<i>values</i>
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=	input block size (default 512)
obs=	output block size (default 512)
bs=	set both input and output block size, superseding <i>ibs</i> and <i>obs</i> ; also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs= <i>n</i>	conversion buffer size
skip= <i>n</i>	skip <i>n</i> input records before starting copy
count= <i>n</i>	copy only <i>n</i> input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
lcase	map alphabetic to lower case
ucase	map alphabetic to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to <i>ibs</i>
... , ...	several conversions separated by commas

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

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

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

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

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

## SEE ALSO

cp(1)

**BUGS**

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. It is not clear how this relates to real life.

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

## NAME

delta — make an SCCS delta

## SYNOPSIS

delta [-s] [-n] [-rsid] [-glist] [-yhistory] [-mmrs] [-p] name ...

## DESCRIPTION

*Delta* adds a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with “s.”), and unreadable files are silently ignored. If a name of “-” is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files, and unreadable files are silently ignored. (If a name of “-” is given the y keyletter must be present; see below.)

A *get* of many SCCS files, followed by a *delta* of those files should be avoided when the *get* generates a large amount of data. Instead, multiple *get-delta* sequences should be used.

Comments about the purpose of the delta(s) are supplied (once, and only once) either from the standard input, or by using the y argument. If one supplies the comments through the standard input, and the standard input is a terminal (as determined by a successful *gety*(II) call), the program will prompt (on the standard output) with “comments? ”. Otherwise, no prompt is printed. A newline preceded by a “\” may be used to make the entering of the comments more convenient. The first newline not preceded by a “\” terminates the comments response. The y argument is used to supply comments on the command line; if it is given the “comments?” question is not printed, and the standard input is not read.

If there is a v flag in the file (see *admin*(I)) the prompting is somewhat different. As the comments are solicited only once, if the first file processed has a v flag then all files processed must have a v flag (any files that don’t will cause a diagnostic message and won’t be processed; processing will continue with the next file). The inverse is also true.

When a file has a v flag, before prompting for “comments? ” *delta* will prompt for “MRs? ” (again, the prompt is only printed if the standard input is a terminal). MR numbers are read from the standard input separated by blanks and/or tabs. The same continuation rules apply as above. When an unadorned newline is read, *delta* will prompt for “comments? ” as described above. If the v flag has a value, it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. This program is executed with the first argument having the value of the %M% identification keyword, a second argument of the value of the %Y% identification keyword, and third and subsequent arguments being the MR numbers. If a non-zero exit status is returned from this program *delta* will terminate (it is assumed that the MR numbers were not all valid). The m argument is used to supply MR numbers on the command line; if it is given the “MRs? ” question is not printed, and the standard input is not read.

The following description is written as though only one SCCS file were named; the process of making a delta is equivalent for each file. (Note that the effects of any keyletter arguments apply independently to each SCCS file, and that the same comments are used for all files.)

The g argument specifies a list (see *get*(I) for the definition of <list>) of deltas which are to be marked *ignored* when the file is accessed at the change level created by this delta. (See the description of the *l-file* format in *get*(I)). A delta should only be ignored when the problem that caused the creation of the delta being ignored is no longer a problem at the change level created

by this delta.

The *p* argument causes *delta* to print the differences that constitute the delta on the standard output.

*Delta* makes a delta by “getting” the named file (see *get(I)*) at the SID specified by the *r* keyletter (this SID *must* be listed in the *p-file*), or at the same SID that was used when the *get* command was executed with the *e* argument by the user executing *delta* (if the user executing *delta* is listed more than once in the *p-file*, the *r* argument *must* be supplied). The “gotten” file is then compared with the *g-file*, the differences between the two files constitute the delta.

When the comparison is finished, *delta* prints the SID of the new delta, followed by the number of lines inserted, deleted, and unchanged. The *s* argument suppresses this printing. Normally, the *g-file* is removed after the delta is made. The *n* argument suppresses the removal.

*Delta* will ignore hangups if it is already ignoring interrupts.

#### FILES

g-file	See <i>get</i> for an explanation of the <i>g-file</i> .
p-file	Information from <i>get</i> .
x-file	Replacement for the SCCS file. The naming convention is the same as that for the <i>p-file</i> (see <i>get</i> ).
z-file	Lockout file; see <i>get(I)</i> .
d-file	“Gotten” file; temporary. The naming convention is the same as that for the <i>p-file</i> (see <i>get</i> ).
/usr/bin/bdiff	Program to compute differences between the “gotten” file and the <i>g-file</i> .

#### SEE ALSO

*get(I)*, *admin(I)*, *prt(I)*, *help(I)*, *scsfile(V)*, *bdiff(I)*  
*SCCS/PWB User's Manual* by L. E. Bonanni and A. L. Glasser.

#### DIAGNOSTICS

Use *help(I)* for explanations.

**NAME**

*deroff* - remove *nroff*, *troff*, and *eqn* constructs

**SYNOPSIS**

**deroff** [ **-w** ] file ...

**DESCRIPTION**

*Deroff* reads each file in sequence and removes all *nroff* and *troff* command lines, backslash constructions, macro definitions, and equations (between “.EQ” and “.EN” lines or between delimiters) and writes the remainder on the standard output. *Deroff* follows chains of included files (“.so” and “.nx” commands); if a file has already been included, a “.so” is ignored and a “.nx” terminates execution. If no input file is given, *deroff* reads from the standard input file.

If the **-w** flag is given, the output is a word list, one “word” (string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed) per line, and all other characters ignored. Otherwise, the output follows the original, with the deletions mentioned above.

**SEE ALSO**

*nroff*(I), *troff*(I), *eqn*(I)

**DIAGNOSTICS**

Complains if a file cannot be opened.

**BUGS**

Does not handle recursive backslash constructions like `\h'\w'c'`.

**NAME**

**df** — report disk free space

**SYNOPSIS**

**df** [**-uqs**] [**-t**number] [arg ...]

**DESCRIPTION**

*Df* prints the number of free blocks on a file system. If no *args* are specified, the free counts of all the mounted file systems are printed.

The **-u** flag prints the total block size, number of blocks allocated for system information, number of free blocks, number of blocks used and the number of free inodes.

The **-q** flag determines and prints the number of free blocks on a file system by extracting the free count directly from the file system's superblock.

The **-s** flag is a silent option which prohibits printing of any results. Error messages and exit status are not effected.

The **-t** flag followed by a decimal *number* (5 digit maximum) is compared with the number of free blocks. The result of the comparison returns the file system's major and minor device numbers and a single character either **Y** or **N**, to indicate if the number of free blocks is greater or less than the requested *number*, respectively (e.g., **df -t 1000 /u8** returns "0 12 Y"). An exit status of 0 is returned for **Y** and 1 for **N**.

The *arg* can be specified as either the root name of the **mounted** file system, e.g., **"/u8"** or the name of the special file corresponding to the particular device (must refer to a disk), e.g., **"/dev/rp14"**.

**FILES**

**/dev/rf?**, **/dev/rk?**, **/dev/rp?**, **/etc/mnttab**

**SEE ALSO**

**ichk(VIII)**

## NAME

diff – differential file comparator

## SYNOPSIS

diff [ -efb ] name1 name2

## DESCRIPTION

*Diff* tells what lines must be changed in two files to bring them into agreement. If *name1* (*name2*) is '-', the standard input is used. If *name1* (*name2*) is a directory, then a file in that directory whose file-name is the same as the file-name of *name2* (*name1*) is used. The normal output contains lines of these forms:

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

These lines resemble *ed* commands to convert file *name1* into file *name2*. The numbers after the letters pertain to file *name2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert file *name2* into *name1*. As in *ed*, identical pairs where  $n1 = n2$  or  $n3 = n4$  are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by '<', then all the lines that are affected in the second file flagged by '>'.

The **-b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal. The **-e** option produces a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate file *name2* from file *name1*. The **-f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file (\$1) and a chain of version-to-version *ed* scripts (\$2,\$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

```
(cat $2 ... $9; echo '1,$p') | ed - $1
```

Except for occasional 'jackpots', *diff* finds a smallest sufficient set of file differences.

## FILES

/tmp/d?????

## SEE ALSO

cmp(I), comm(I), ed(I), uniq(I)

## DIAGNOSTICS

'jackpot' – To speed things up, the program uses hashing. You have stumbled on a case where there is a chance that this has resulted in a difference being called where none actually existed. Sometimes reversing the order of files will make a jackpot go away.

## BUGS

Editing scripts produced under the **-e** or **-f** options are naive about creating lines consisting of a single '.'.



**NAME**

`diff3` - 3-way differential file comparison

**SYNOPSIS**

`diff3` [ `-ex3` ] *file1* *file2* *file3*

**DESCRIPTION**

*Diff3* compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

```

=====    all three files differ
=====1    file1 is different
=====2    file2 is different
=====3    file3 is different

```

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

```

f: n1 a    Text is to be appended after line number n1 in file f, where f= 1, 2, or 3.
f: n1,n2 c  Text is to be changed in the range line n1 to line n2. If n1 = n2, the range may
                be abbreviated to n1.

```

The original contents of the range follows immediately after a `c` indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the `-e` option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e. the changes that normally would be flagged `=====` and `=====3`. Option `-x` (`-3`) produces a script to incorporate only changes flagged `=====` (`=====3`). The following command will apply the resulting script to 'file1'.

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

**SEE ALSO**

`diff(1)`

**BUGS**

Text lines that consist of a single '.' will defeat `-e`.

**FILES**

```

/tmp/d3a?????
/tmp/d3b?????

```

## NAME

diffmark – mark changes between versions of a file

## SYNOPSIS

**diffmark** [*code*"string"] ... [*name*]

## DESCRIPTION

*Diffmark* is a filter used to modify the editor command version of *diff(1)* output so that it can be used to mark the changes between successive versions of a file. Its most common use is to automatically insert change mark commands into a file of text for *nroff(1)* or *troff(1)*. The following is a typical command sequence:

```
diff -e oldfile newfile | diffmark markedfile | ed - oldfile
```

The generated file *markedfile* is the same as *newfile*, except that it has the needed change mark requests inserted. The user would normally print *markedfile*, and later remove it and *oldfile* when no longer needed.

*Diffmark* adds extra lines to the output of *diff*. It inserts one line at both the beginning and end of each sequence of appended or changed lines, and appends two lines following each deletion. The default values of these lines are chosen to make use of the "margin character" request of the formatters. The user may override any such value by supplying an option string, which is concatenated with a newline to make up the line. Any null option string causes its corresponding line to be omitted completely. The option codes and their defaults are as follows:

–**ab**".mc|" – "append" beginning – insert at beginning of an addition

–**ae**".mc" – "append" end – insert at end of an addition

–**cb**".mc|" – "change" beginning – insert at beginning of a change

–**ce**".mc" – "change" end – insert at end of a change

–**db**".mc\*" – "delete" 1st line – insert as first line of deletion

–**de**".mc" – "delete" 2nd line – insert as 2nd line of deletion

Although not a necessity, the following option is convenient:

*name* causes *diffmark* to append "w *name*" to the end of its output. For safety's sake, this name should *not* be the same as that of the file being edited.

Here is an example. Suppose you run the following sequence of commands:

```
diff -e oldfile newfile >diff1
diffmark diff3 -cb".mc +" <diff1 >diff2
ed - oldfile <diff2
nroff diff3 >diff4
```

Of course, the only reason for doing this rather than using pipelines is to see what all the files look like:

oldfile:

.nf  
ccc  
eee  
ggg  
hhh  
zzz

newfile:

.nf  
aaa  
eee  
fff  
ggg  
zzz

diff1 (output from diff):

5d  
3a  
fff  
.  
2c  
aaa  
.

diff2 (output from diffmark):

5c  
.mc \*  
.mc  
.  
3a  
.mc |  
fff  
.mc  
.  
2c  
.mc +  
aaa  
.mc  
.

w diff3

diff3 (edited version of oldfile):

.nf  
.mc +  
aaa  
.mc  
eee  
.mc |  
fff  
.mc  
ggg

```
.mc *
.mc
zzz
```

diff4 (formatted output, with line length set to 10):

```
aaa      +
eee
fff      |
ggg
zzz      *
```

If you are so inclined, you can use *diffmark* to produce listings of C (or other) programs with changes marked. A typical shell procedure is:

```
:          cdiffmk: shell proc to show C program differences
:          called: cdiffmk old new
diff -e $1 $2 | (diffmark;echo '1,$p') | ed - $1 | nroff macs - | pr -h $2
```

The file macs looks like this:

```
.pl1
.ll 77
.nf
.eo
.nc
```

The ll request might specify a different line length, depending on the nature of the program being printed. The eo and nc requests are probably needed only for C programs.

#### DIAGNOSTICS

"input not from diff"

"line too long" (>512 characters)

Up to 72 characters of the offending line are printed immediately following the diagnostic.

#### EXIT CODES

0 - normal completion

1 - input did not appear to be from *diff*, or other error

#### SEE ALSO

diff(1), nroff(1), troff(1)

#### BUGS

Esthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output. I.e., replacing ".sp" by ".sp2" will produce a change mark on the preceding or following line of output.

For those who use *diffmark* to produce UNIX Manual pages, extra handling may be needed to get vertical bars to appear. This results from the choice of the bar as the character translated to a nonadjustable blank for use with tabs. When you use *diffmark*, override the default choice of "|" by "!" instead, causing the latter to appear in your final output. If you prefer the vertical bar, you can get it on the final output by adding the following to the beginning of your file:

```
.if n .tr !|
.if n .ds v !
```

which may be mysterious, but works.

**NAME**

*dsw* - delete interactively

**SYNOPSIS**

*dsw* [ directory ]

**DESCRIPTION**

For each file in the given directory ( '.' if not specified) *dsw* types its name. If *y* is typed, the file is deleted; if *x*, *dsw* exits; if new-line, the file is not deleted; if anything else, *dsw* asks again.

**SEE ALSO**

*rm*(1)

**BUGS**

The name *dsw* is a carryover from the ancient past. Its etymology is amusing.

**NAME**

`du` - summarize disk usage

**SYNOPSIS**

`du [ -s ] [ -a ] [ name ... ]`

**DESCRIPTION**

*Du* gives the number of blocks contained in all files and (recursively) directories within each specified directory or file *name*. If *name* is missing, '.' is used.

The optional argument `-s` causes only the grand total to be given. The optional argument `-a` causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

A file which has two links to it is only counted once.

**BUGS**

Non-directories given as arguments (not under `-a` option) are not listed.

Removable file systems do not work correctly since i-numbers may be repeated while the corresponding files are distinct. *Du* should maintain an i-number list per root directory encountered.

**NAME**

echo — echo arguments

**SYNOPSIS**

**echo** [ arg ... ]

**DESCRIPTION**

*Echo* writes its arguments in order as a line on the standard output file. It is mainly useful for producing diagnostics in command files.

Certain escape sequences are recognized:

“\n” causes the newline character to be written.

“\c” terminates *echo* without a newline.

“\0N” causes the octal number *N* to be written.

**SEE ALSO**

pump(I)

## NAME

ed — text editor

## SYNOPSIS

ed [ - ] [ + ] [ name ]

## DESCRIPTION

*Ed* is the standard text editor.

If the *name* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional '-' suppresses the printing of character counts by *e*, *r*, and *w* (or *z*) commands.

*Ed* operates on a copy of the file it is editing; changes made in the copy have no effect on the file until a *w* or *z* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

If changes have been made in the buffer since the last *w* or *z* command, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *q* or *e* commands. *Ed* prints 'q?' or 'e?', respectively, and allows one to continue editing. A second *q* or *e* command at this point will take effect. This warning feature may be inhibited by specifying the '+' option (e.g., *ed + file*). The '-' option also inhibits this feature.

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 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.

*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 replaced. 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. The regular expressions allowed by *ed* are constructed as follows:

The following *one-character regular expressions* match a single character:

- 1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character regular expression that matches itself.
- 1.2 A backslash '\' followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:
  - a. '.', '\*', '+', '[', and '\' (period, asterisk, plus sign, left square bracket, and backslash, respectively), which are always special, except when they appear within square brackets '[' (see 1.4 below).
  - b. '^' (caret or circumflex), which is special at the beginning of an *entire regular expression* (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets '[' (see 1.4 below).
  - c. '\$' (currency symbol), which special at the end of an *entire regular expression* (see 3.2 below).



- d. The character used to bound (i.e., delimit) an entire regular expression, which is special for that regular expression (for example, see how '/' is used in the *g* command, below.)
- 1.3 A period '.' is a one-character regular expression that matches any character except the new-line character.
  - 1.4 A non-empty string of characters enclosed in square brackets '[' is a one-character regular expression that matches *any one* character in that string. If, however, the first character of the string is a circumflex '^', the one-character regular expression matches any character *except* 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 ']' does not terminate such a string when it occurs first (after an initial '^', if any), in it, e.g., '[a]' matches either a right square bracket ']' or the letter 'a'. The five characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *regular expressions* from *one-character regular expressions*:

- 2.1 A one-character regular expression is a regular expression that matches whatever the one-character regular expression matches.
- 2.2 A one-character regular expression followed by an asterisk '\*' is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, this regular expression matches as many occurrences as possible.
- 2.3 A one-character regular expression followed by a plus '+' is a regular expression that matches *one* or more occurrences of the one-character regular expression. If there is any choice, this regular expression matches as many occurrences as possible.
- 2.4 A one-character regular expression followed by '\{ m \}', '\{ m, \}', or '\{ m, n \}' is a regular expression that matches a *range* of occurrences of the one-character regular expression. The values of *m* and *n* must be non-negative integers less than 256; '\{ m \}' matches exactly *m* occurrences; '\{ m, \}' matches at least *m* occurrences; '\{ m, n \}' matches any number of occurrences between *m* and *n* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.
- 2.5 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.
- 2.6 A regular expression enclosed between the character sequences '\(' and '\)' is a regular expression that matches whatever the unadorned regular expression matches; this construction has side effects discussed under the *s* command, below.

Finally, an *entire regular expression* may be constrained to match only an initial segment or final segment of a line (or both):

- 3.1 A circumflex '^' at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.
- 3.2 A currency symbol '\$' at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line. The construction *^entire regular expression\$* constrains the entire regular expression to match the entire line.

The null regular expression standing alone (e.g., '/') is equivalent to the last regular expression encountered.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of the command. Addresses are constructed as follows:

1. The character '.' addresses the current line.
2. The character '\$' addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. 'x' addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A regular expression 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 regular expression. If necessary, the search wraps around to the beginning of the buffer and continues through the current line, so that the entire buffer is searched.
6. A regular expression enclosed in queries '?' 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 regular expression. If necessary the search wraps around to the end of the buffer and continues through the current line.
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 or subtracted, respectively. As a consequence of this rule and of rule 8, the address '-' refers to the line preceding the current line. Moreover, trailing '+' and '-' characters have a cumulative effect, so '---' refers to the current line less 2.
10. To maintain compatibility with earlier versions of the editor, the character '^' in addresses is entirely equivalent to '-'.

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) 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 before the second address is interpreted. This feature can be used to determine the starting line for forward and backward searches (see items 5. and 6. in the list 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, but are used to show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed, respectively, as discussed below under the *p* and *l* 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: text is placed at the beginning of the buffer.

(.,.)c  
<text>

The *change* command deletes the addressed lines, then accepts input text which replaces these lines. '.' is left at the last line input; if there were none, it is left at the first line 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 name

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 remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *name* is remembered for possible use as a default file name in subsequent *e* or *r* or *w* or *z* commands.

f name

If *name* is given, the *filename* command changes the currently remembered file name to *name*; otherwise, it prints the currently remembered file name.

(1,S)g/regular expression/command list

In the global command, the first step is to mark every line that matches the given *regular expression*. Then, for every such line, the given *command list* is executed with '.' initially set to that line. A single command or the first of 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*. The (global) commands (*g*, *v*, *G*, and *V*) are *not* permitted in the *command list*.

(.)h

The date as returned by *date*(1) is appended after the addressed line.

(.)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.

(.,.+1)j

The *join* command joins contiguous lines by removing the appropriate new-line characters.

(.)kx

The *mark* command marks the addressed line with name *x*, which must be a lower-case letter. The address form '*x*' then addresses this line.

(. . .)l

The *l* command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab*, *backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may also be appended to any other command.

(. . .)ma

The *move* command repositions the addressed line(s) after the line addressed by *a*. Address '0' is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines. The last line so moved becomes the current line.

(. . .)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 (e.g., '*dp*' deletes the current line and prints the new current line).

q

The *quit* command causes *ed* to exit. No automatic write of a file is done.

(\$)r name

The *read* command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). The remembered file name is not changed unless *name* 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.

(. . .)s/regular expression/replacement/ or,

(. . .)s/regular expression/replacement/g

The *substitute* command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all (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. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of '/' to delimit the regular expression and the *replacement*; '.' is left at the last line on which a substitution occurred.

An ampersand '&' appearing in the *replacement* is replaced by the string matching the regular expression 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 regular expression enclosed between '\(' and '\)'. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

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 by '\'. Such substitution cannot be done as part of a *g* or *v* command list.

(. . .)ta

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be '0'); '.' is left at the last line of the copy.

u

This command reverses the effect of the last *s* command. The *u* command affects only the last line changed by the most recent *s* command.

## (1,\$) v/regular expression/command list

This command is the same as the global command *g* except that the *command list* is executed with '.' initially set to every line that does *not* match the *regular expression*.

## (1,\$) w name

## (1,\$) z name

The write command writes the addressed lines onto the named file. If the file does not exist, it is created with mode 644 (readable by everyone, writable by you). The remembered file name is *not* changed unless *name* is the very first file name mentioned since *ed* was invoked. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands); '.' is unchanged. If the command is successful, the number of characters written is typed. The *z* command is identical to *w* but, on most keyboards, the 'z' key is farther from the 'q' key than is the 'w' key.

## (1,\$) G/regular expression/

In the interactive Global command, the first step is to mark every line that matches the given *regular expression*. Then, for every such line, that line is printed, '.' is changed to that line, and any *one* command, other than a global command (*g*, *v*, *G*, and *V*), must be input. After the execution of that command, the next marked line is printed, and so on. A new-line acts as a null command; an '&' causes the re-execution of the most recent command executed within this invocation of *G*. Note that the commands input after the *G* command prints each marked line may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

## P

The editor will prompt with a '\*' for all subsequent commands. This command alternately turns the mode on and off; it is initially off.

## Q

The editor exits without checking if changes have been made in the buffer since the last *w* or *z* command.

## (1,\$) V/regular expression/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the *regular expression*.

## (\$) =

The line number of the addressed line is typed; '.' is unchanged by this command.

## !UNIX command

The remainder of the line after the '!' is sent to the UNIX shell (*sh*(1)) to be interpreted as a command; '.' is unchanged.

## (.+1) &lt;new-line&gt;

An address alone on a line causes the addressed line to be printed. A blank line alone is equivalent to '+1p'; it is useful for stepping through text.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a '?' and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

*Ed* allows the user to include, in the first line of each text file, a specification to control the line length and the tab-to-space conversion. For example, <:t5,10,15s72:> sets tabs at columns 5, 10, and 15; it will also truncate the *printing* of all lines to a length of 72 characters and warn when a truncation has occurred. For the specification to take effect, the user's terminal must be in *echo*

and *-tabs* modes (see *str*(1)). Only the 't' and 's' parameters may be used as described in *fspec*(V). If the 's' parameter is used, all referenced lines are checked for maximum length on file read and write operations and on line print operations. Appropriate diagnostics are generated. Truncation occurs *only* on printing.

If the user attempts a *w* or *z* command and the destination file system does not have enough space available, a diagnostic is printed with an error number (i.e. "NO SPACE: e1"). *Ed* will not perform the write. The *UNIX* command "help e1" (see *help*(1)) prints out a full description of what to do. *Help* should be executed before leaving the editor (e.g., "!help e1").

#### FILES

/tmp/e#, temporary; '#' is the process number (in octal).

#### DIAGNOSTICS

'?' for errors in commands; 'TMP?' for temporary file (buffer) overflow; *help*(1) error numbers in all other cases. Commands in error should be re-entered properly. On temporary file overflow, the buffer should be written to a file and then an *e* command executed on that file. This will re-initialize the buffer; note that if the buffer overflows during the execution of a command that, in the absence of the TMP? diagnostic, would have done several changes, only some of the changes may have been done. Help error messages are self-explanatory.

#### SEE ALSO

*A Tutorial Introduction to the UNIX Text Editor* by B. W. Kernighan.  
*Advanced Editing on UNIX* by B. W. Kernighan.

#### BUGS

If the *s* command succeeds on (i.e., modifies) a line that was marked by a *g*, *v*, *G*, or *V* command, then that mark is effectively removed. The editor deletes all ASCII *null* characters whenever it reads text into the buffer.

## NAME

`egrep` - search a file for lines containing a pattern

## SYNOPSIS

`egrep` [ `-b` ] [ `-c` ] [ `-f` ] [ `-n` ] [ `-v` ] *pattern* [ *file* ] ...

## DESCRIPTION

*egrep* searches the input files (standard input default) for all lines containing an instance of the regular expression *pattern*. Normally, each line matched is copied to the standard output. The *pattern* matches a line whenever the line contains a substring denoted by the *pattern*.

The flags modify the normal behavior as follows:

- `-b` causes each printed line to be preceded by the block number on which it was found
- `-c` causes only a count of matching lines to be printed
- `-f` causes the regular expression to come from a file named *pattern*
- `-n` causes each printed line to be preceded by its relative line number in the file
- `-v` causes all lines but those matching the *pattern* to be printed

In all cases the file name is shown if there is more than one input file.

A *pattern* is one of the following:

1. an ordinary character (denoting itself)
2. a circumflex '^' (denoting the beginning of a line)
3. a dollar sign '\$' (denoting the end of a line)
4. a period '.' (denoting any character but a newline)
5. '[' followed by a string of characters followed by ']' (denoting any character in the string; if the first character in this string is '^', the pattern denotes any character except newline and the characters in the string)
6. '(' followed by a pattern followed by ')' (denoting the enclosed pattern)
7. a pattern followed by '\*', or by '+', or by '?' (denoting zero or more, one or more, or zero or one instances, respectively, of the preceding pattern)
8. a pattern followed by a pattern (denoting concatenation of the two patterns)
9. a pattern followed by '|' followed by another pattern (denoting the alternation of the two patterns); a newline may be used in place of '|'.

In parsing a pattern, the rules are applied in the order given.

A pattern metacharacter can be used as an ordinary character by preceding it by '\'. The metacharacters are: '^', '\$', '.', '[', ']', '\*', '+', '?', '(', ')', '\\'.

Care should be taken when using the characters \$ \* [ ^ | ( ) and \ in the regular expression as they are also meaningful to the Shell. When *pattern* is a regular expression other than a simple string, it is generally necessary to enclose the entire *pattern* argument in quotes.

## SEE ALSO

`grep`(1), `fgrep`(1), `lex`(1), `rgrep`(1), `sed`(1), `ed`(1), `sh`(1)

## BUGS

Lines longer than 512 characters are not printed completely.

## NAME

eqn — typeset mathematics

## SYNOPSIS

eqn [ file ] ...

## DESCRIPTION

*Eqn* is a *troff*(1) preprocessor for typesetting mathematics on the Graphics Systems, Inc. phototypesetter. Usage is almost always

eqn file ... | troff

If no files are specified, *eqn* reads from the standard input. A line beginning with “.EQ” marks the start of an equation; the end of an equation is marked by a line beginning with “.EN”. Neither of these lines is altered or defined by *eqn*, so you can define them yourself in *troff*(1) to get centering, numbering, etc. All other lines are treated as comments, and passed through untouched.

Spaces, tabs, new-lines, braces, double quotes, tilde, and circumflex are the only delimiters. Braces “{}” are used for grouping. Use tildes “~” to get extra spaces in an equation.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub i* makes  $x_i$ , *a sub i sup 2* produces  $a_i^2$ , and *e sup {x sup 2 + y sup 2}* gives  $e^{x^2+y^2}$ . Fractions are made with **over**. *a over b* is  $\frac{a}{b}$  and *1 over sqrt {ax sup 2 + bx + c}* is  $\frac{1}{\sqrt{ax^2+bx+c}}$ ; **sqrt** makes square roots.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things:  $\lim_{n \rightarrow \infty} \sum_0^n x_i$  is made with *lim from {n-> inf} sum from 0 to n x sub i*. Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [ x sup 2 + y sup 2 over alpha right ] ~ = 1* produces  $\left[ x^2 + \frac{y^2}{\alpha} \right] = 1$ . The **right** clause is optional.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces  $\begin{matrix} a \\ b \\ c \end{matrix}$ . There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **bar**: *x hat = f(t) bar* is  $\hat{x} = \overline{f(t)}$ . Default sizes and fonts can be changed with **size n** and various of **roman**, **italic**, and **bold**.

Keywords like *sum* ( $\Sigma$ ), *int* ( $\int$ ), *inf* ( $\infty$ ), and shorthands like  $>=$ ,  $(\geq)$ ,  $->$ ,  $(\rightarrow)$ ,  $!=$ ,  $(\neq)$  are recognized. Spell out Greek letters in the desired case, as in *alpha*, *GAMMA*. Mathematical words like *sin*, *cos*, *log* are made Roman automatically. *Troff*(1) four-character escapes like  $\backslash(ua$  ( $\uparrow$  — for “up arrow”) can be used anywhere. Strings enclosed in double quotes “...” are passed through untouched.

## SEE ALSO

*Typesetting Mathematics — User's Guide (2nd Edition)* by B. W. Kernighan and L. L. Cherry  
*New Graphic Symbols for EQN and NEQN* by C. Scrocca  
*NROFF/TROFF User's Manual* by J. F. Ossanna  
*troff*(1), *neqn*(1)



**BUGS**

Undoubtedly. Watch out for small or large point sizes — it's tuned too well for size 10. Be cautious if inserting horizontal or vertical motions, and of backslashes in general.

## NAME

= (equals) – shell assignment command

## SYNOPSIS

= letter [ arg1 [ arg2 ] ]

## DESCRIPTION

The “=” command provides shell string variables. The 26 *letter* variables (‘a’–‘z’) are referenced in later commands in the manner of shell arguments, i.e.: \$a, ..., \$z. If no arguments are given, the standard input is read to newline or EOT for the value. The exit code is set to 0 if a newline is found in the input; it is set to 1 otherwise, thus providing an end-of-file indicator. If *arg1* is the only argument, or if two non-null arguments are given, the variable is set to *arg1*, and the exit code set to 0. If two arguments are given, and if *arg1* is a null string, the value of *arg2* is assigned to the variable, and the exit code is set to 1, permitting a convenient default mechanism:

= a "\$1" "default value" && shift

The “=” command works either at the terminal, or in shell command files. The variables can be assigned repeatedly. Storage is assigned as needed, but there is no recovery.

Some *letter* variables have predefined meanings and are initialized once at the time the Shell begins execution:

\$n The argument count. “sh file arg1 arg2 arg3” has the value 3. The shift command does not change the value of \$n.

\$p This variable holds the shell search sequence of pathname prefixes for command execution. Alternatives are separated by “:”. The default initial value is:

= p ":/bin:/usr/bin"

which prepends successively

the null pathname (execute from current dir.),

/bin,

/usr/bin.

Using the same type of specification, users may choose their own sequence by storing it in a file named “.path” in their login directory. The “.path” information passes to successive shells (and other commands like *time(1)* or *nohup(1)*); the \$p value does not. In any case, no prepending occurs when a command name contains a ‘/’.

\$r exit(status) of the most recent command executed by the Shell. The value is ASCII numeric, and is initially ‘0’. At end-of-file the shell exits with the value of \$r.

\$s Name of login (starting) directory.

\$t Terminal identification letter or number: /dev/tty\$t is a file name for the terminal.

\$w First component in \$s pathname, i.e., file system name (such as /usr).

\$z Is the name of the Shell. Its default value is ‘/bin/sh’, but this can be overridden by supplying a name as the second line of the ‘.path’ file.

Note that variables (‘a’ – ‘m’) are guaranteed to be initialized to null strings and usable in any way desired. Variables (‘n’ – ‘z’) may acquire special uses in the future. The values of \$n, \$s, \$t, and \$w may reasonably be modified; it is catastrophic to change \$p; it is possible, but useless to modify \$r.

The “=” command is executed within the shell. Note that it is commonly used to read the first line of output from a pipe or a line from the terminal, for example:

```
grep -c string file | = a
```

or:

```
= a </dev/tty
```

#### EXIT CODES

0 – normal read, or first of two arguments is not null.

1 – end-of-file, or first of two arguments is null.

#### SEE ALSO

expr(I), sh(I)

**NAME**

`exit` -- terminate command file

**SYNOPSIS**

`exit` [ integer ]

**DESCRIPTION**

*Exit* performs a `seek` to the end of its standard input file. Thus, if it is invoked inside a file of commands, upon return from *exit* the shell will discover an end-of-file and terminate.

The optional argument will be returned to the shell as the exit status code.

**SEE ALSO**

`if(1)`, `goto(1)`, `sh(1)`

**NAME**

*expr* — evaluate arguments as an algebraic expression

**SYNOPSIS**

**expr** *arg ...*

**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, i.e., '\*', '|', '&', '(', and ')', must be escaped.

The operators and keywords are listed below. Characters that need to be escaped are preceded by '\'. The list is in order of increasing precedence, with equal precedence operators grouped within '{}' symbols.

*expr* \| *expr*

*expr* \& *expr*

*expr* { +, - } *expr*

*expr* { \\*, /, % } *expr*

**substr** *expr expr expr*

**length** *expr*

**index** *expr expr*

\( *expr* \)

The result of *substr* is that portion of the first expression (possibly null) which is defined by the offset (second expression, starting at '1') and the span (third expression). A large span value can be given to obtain the remainder of the string.

*Length* returns the length in characters of the expression that follows.

*Index* searches the first expression for the first character that matches a character from the second expression. It returns the character position number if it succeeds, or '0' if it fails.

The *expr* command is handy with Shell variables. For example:

```
expr substr xxx$a "(" length xxx$a - 2 ")" 3 | = b
```

assigns the last three characters of the Shell variable *\$a* into the variable *\$b*.

Note that '0' is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted.

**DIAGNOSTICS**

Grumbles from *yacc*(1) for syntax violations.

"Non-numeric argument" if the argument needs to be, but is not, an integer.

## NAME

`fc` – Fortran compiler

## SYNOPSIS

`fc [ -c ] sfile1.f ... ofile1 ...`

## DESCRIPTION

`Fc` is the UNIX Fortran compiler. It accepts three types of arguments:

Arguments whose names end with `.f` are assumed to be Fortran source program files; they are compiled, and the object program is left on the file `sfile1.o` (i.e., the file whose name is that of the source with `.o` substituted for `.f`).

Other arguments (except for `-c`) are assumed to be either loader flags, or object programs, typically produced by an earlier `fc` run, or perhaps libraries of Fortran-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name **a.out**.

The `-c` argument suppresses the loading phase, as does any syntax error in any of the routines being compiled.

The following is a list of differences between `fc` and ANSI standard Fortran (also see the BUGS section):

1. Arbitrary combination of types is allowed in expressions. Not all combinations are expected to be supported at runtime. All of the normal conversions involving integer, real, double precision and complex are allowed.
2. Two forms of “implicit” statements are recognized: **implicit integer /i-n/** or **implicit integer (i-n)**.
3. The types **doublecomplex**, **logical\*1**, **integer\*1**, **integer\*2**, **integer\*4** (same as **integer**), **real\*4 (real)**, and **real\*8 (double precision)** are supported.
4. **&** as the first character of a line signals a continuation card.
5. **c** as the first character of a line signals a comment.
6. All keywords are recognized in lower case.
7. The notion of ‘column 7’ is not implemented.
8. G-format input is free form; leading blanks are ignored, the first blank after the start of the number terminates the field.
9. A comma in any numeric or logical input field terminates the field.
10. There is no carriage control on output.
11. A sequence of *n* characters in double quotes “” is equivalent to *n* **h** followed by those characters.
12. In **data** statements, a hollerith string may initialize an array or a sequence of array elements.
13. The number of storage units requested by a binary **read** must be identical to the number contained in the record being read.
14. If the first character in an input file is “#”, a preprocessor which implements “#define” and “#include” preprocessor statements is called. These preprocessor statements are similar to the corresponding C preprocessor statements; see the C reference manual for details. The

preprocessor does not recognize Hollerith strings written with *nh*.

In I/O statements, only unit numbers 0-19 are supported. Unit number *n* refers to file *fortnn*; (e.g. unit 9 is file 'fort09'). For input, the file must exist; for output, it will be created. Unit 5 is permanently associated with the standard input file; unit 6 with the standard output file. Also see *setfil*(III) for a way to associate unit numbers with named files.

#### FILES

a.out	loaded output
f.tmp[123]	temporary (deleted)
/usr/fort/errors	compile-time error messages
/usr/fort/fcl	compiler proper
/lib/fr0.o	runtime startoff
/lib/filib.a	interpreter library
/lib/libf.a	builtin functions, etc.
/lib/liba.a	system library

#### SEE ALSO

*rc*(I), which announces a pleasant Fortran dialect; the ANSI standard; *ld*(I) for loader flags. For some subroutines, try *ierror*, *getarg*, *setfil*(III).

#### DIAGNOSTICS

Compile-time diagnostics are given in English, accompanied if possible with the offending line number and source line with an underscore where the error occurred. Runtime diagnostics are given by number as follows:

1	invalid log argument
2	bad arg count to amod
3	bad arg count to atan2
4	excessive argument to cabs
5	exp too large in cexp
6	bad arg count to cmplx
7	bad arg count to dim
8	excessive argument to exp
9	bad arg count to idim
10	bad arg count to isign
11	bad arg count to mod
12	bad arg count to sign
13	illegal argument to sqrt
14	assigned/computed goto out of range
15	subscript out of range
16	real**real overflow
17	(negative real)**real
100	illegal I/O unit number
101	inconsistent use of I/O unit
102	cannot create output file
103	cannot open input file
104	EOF on input file
105	illegal character in format
106	format does not begin with (
107	no conversion in format but non-empty list
108	excessive parenthesis depth in format
109	illegal format specification

- 110 illegal character in input field
- 111 end of format in hollerith specification
- 112 bad argument to setfil
- 120 bad argument to ierror
- 999 unimplemented input conversion

**BUGS**

The following is a list of those features not yet implemented:  
arithmetic statement functions  
scale factors on input  
**Backspace** statement.



**NAME**

**fd2** — redirect file descriptor 2 (diagnostic output)

**SYNOPSIS**

**fd2** [ *fd2arg* ] *command* [ *command-arg* ] ...

**DESCRIPTION**

*Fd2* executes *command* with file descriptor 2 (the diagnostic output) redirected to a file or to the standard output. There are three forms:

<b>fd2</b> <b>-file</b> <i>comd</i> ...	[write on <i>file</i> ]
<b>fd2</b> <b>--file</b> <i>comd</i> ...	[append to <i>file</i> ]
<b>fd2</b> <b>+</b> <i>comd</i> ...	[causes file descriptor 2 to be made the same as file descriptor 1]

In either of the first two cases, omission of *file* causes **msg.out** to be used as the output file. Omission of *fd2arg* has the effect of **-msg.out**.

**NAME**

`fgrep` – search a file for lines containing keywords

**SYNOPSIS**

`fgrep [ -b ] [ -c ] [ -e ] [ -f ] [ -n ] [ -v ] pattern [ file ] ...`

**DESCRIPTION**

*fgrep* searches the input files (standard input default) for all lines containing one or more keywords denoted by *pattern*. Normally, each containing line is copied to the standard output. The **bcfnv** flags modify the normal behavior as in *egrep*(1). The **-e** flag causes a match to occur if and only if a keyword matches an input line exactly. Without the **-f** flag, *pattern* can be only a single keyword. With the **-f** flag, *pattern* is the name of a file containing a sequence of keywords terminated by newlines. The keywords in this file then constitute the search pattern. A keyword is any string of characters except `'\0'` and newline. No metacharacters are assumed.

**SEE ALSO**

`egrep`(1)

**BUGS**

Lines longer than 512 characters are not printed completely.

**NAME**

**file** - determine file type

**SYNOPSIS**

**file** file ...

**DESCRIPTION**

*File* performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ascii, *file* examines the first 512 bytes and tries to guess its language.

## NAME

find - find files

## SYNOPSIS

find pathname-list expression

## DESCRIPTION

*Find* recursively descends the directory hierarchy for each pathname in the *pathname-list* (i.e., one or more pathnames) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where  $+n$  means more than *n*,  $-n$  means less than *n* and *n* means exactly *n*.

- name** filename True if the *filename* argument matches the current file name. Normal Shell argument syntax may be used if escaped (watch out for '[', '?' and '\*').
- perm** onum True if the file permission flags exactly match the octal number *onum* (see *chmod*(I)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(II)) become significant and the flags are compared:  $(flags \& onum) == onum$ .
- type** c True if the type of the file is *c*, where *c* is **b**, **c**, **d** or **f** for block special file, character special file, directory or plain file.
- links** n True if the file has *n* links.
- user** uname True if the file belongs to the user *uname*.
- group** gname As it is for **-user** so shall it be for **-group** (someday).
- size** n True if the file is *n* blocks long (512 bytes per block).
- atime** n True if the file has been accessed in *n* days.
- mtime** n True if the file has been modified in *n* days.
- exec** command True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument '{' is replaced by the current pathname.
- ok** command Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds **y**.
- print** Always true; causes the current pathname to be printed.

The primaries may be combined with these operators (ordered by precedence):

- !** Prefix *not*.
- a** Infix *and*, second operand evaluated only if first is true.
- o** Infix *or*, second operand evaluated only if first is false.
- ( expression )** Parentheses for grouping. (Must be escaped.)

To remove all files named 'a.out' or '\*.o' that have not been accessed for a week:

```
find / "(" -name a.out -o -name "*.o" ")" -a -atime +7 -a -exec rm {} ";"
```

## FILES

/etc/passwd

**SEE ALSO**

sh(1), if(1), fs(V)

**BUGS**

*Test* (see *if(1)*) can be useful with *find*. However, since *test* is implemented within the Shell, you must use something like:

```
-exec sh -c "test args" ";"
```

## NAME

*gath* — gather real and virtual files

## SYNOPSIS

***gath*** [**-ih**] file ...

## DESCRIPTION

*Gath* concatenates the named files and writes them to standard output. Tabs are expanded into spaces according to the format specification for each file (see *fspec(V)*). The size limit and margin parameters of a format specification are also respected. Non-graphic characters other than tabs are identified by a diagnostic message and excised. The output of *gath* contains no tabs unless the **-h** flag is set, in which case it is written with standard tabs (every eighth column).

Any line of any of the files which begins with '~' is interpreted by *gath* as a control line. A line beginning '~ ' (tilde,space) specifies a sequence of files to be included at that point. A line beginning '~!' specifies a UNIX command; that command is executed, and its output replaces the '~!' line in the *gath* output.

Setting the **-i** flag prevents control lines from being interpreted and causes them to be output literally.

A file name of '-' at any point refers to standard input, and a control line consisting of '~.' is a software end-of-file. Keywords may be defined by specifying a replacement string which is to be substituted for each occurrence of the keyword. Input may be collected directly from the terminal, with several alternatives for prompting.

In fact, all of the special arguments and flags recognized by the *send* command are also recognized and treated identically by *gath*. Several of them are only useful, however, in the context where an RJE job is being submitted. The same program implements the two commands, so *gath* has a potential which is not apparent from its name. Refer to the description of *send* for definitive information about *gath*.

## SEE ALSO

*send(I)*, *fspec(V)*

## NAME

get — get generation from SCCS file

## SYNOPSIS

```
get [-rrel[.lev[.br[.seq]]]] [-ccutoff] [-iincl-list] [-xexcl-list] [-aserial] [-k] [-e] [-l[p]]
[-p] [-m] [-n] [-s] [-b] [-g] [-t] name ...
```

## DESCRIPTION

*Get* generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with “-”. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with “s.”), and unreadable files are silently ignored. If a name of “-” is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files, and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file*. See FILES, below, for an explanation of how the name of this file is determined.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

- r The SCCS identification string (SID) of the change level to be generated. If the entire argument is omitted, the meaning is the same as if the default SID were specified (see *admin(1)*). If there is no default SID in the SCCS file the highest release which has deltas is used. If only the release is specified, the level defaults to the highest level in that release. A release and level completely identifies a specific change level. If a branch is also specified and the sequence is omitted, the sequence defaults to the highest sequence in the branch. A release, level, branch, and sequence also completely identifies a specific change level. (All deltas are identified either by a 2 component SID—release and level, or by a 4 component SID—release, level, branch, and sequence. SID’s with 4 components identify deltas which have heretofore been called “non-propagating”.)
- c Cutoff date-time, in the form YY[MM[DD[HH[MM[SS]]]]]. No delta which was created after the specified cutoff date-time will be applied. Units omitted from the date-time default to their maximum possible values; that is, “-c7502” is equivalent to “-c750228235959”. Any number of non-numeric characters may separate the various 2 digit pieces of the cutoff date-time. This feature allows one to specify a cutoff date in the form: “-c77/2/2 9:22:25”. Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send(1)* command:
 

```
~!get "-c%E% %U%" s.file
```
- i This argument is used to specify a list of deltas to be included (forced to be applied). The list has the following syntax:

```
<list> ::= <range>
          | <list> , <range>
<range> ::= <delta>
```

```

| <delta> - <delta>
<delta> ::= <rel>
| <rel> . <lev>
| <rel> . <lev> . <br>
| <rel> . <lev> . <br> . <seq>

```

If a level is omitted from a delta specification the highest level of the specified release is assumed. If a branch is specified, but the sequence is omitted the highest sequence of the specified branch is assumed.

- x This argument is similar to *i* except that it is followed by a list of deltas to be excluded (forced to not be applied).
- a The serial number of the change level to be generated (see *sccsfile(V)*). This keyletter is used by the *comb(I)* command; it is not a generally useful keyletter, and most users will probably never use it. If both the *r* and *a* keyletters are specified, the *a* keyletter is used. Care should be taken when using the *a* keyletter in conjunction with the *e* keyletter, as the SID of the delta to be created may not be what one expects. The *r* keyletter can be used with the *a* and *e* keyletters to control the naming of the SID of the delta to be created.
- k This argument suppresses replacement of identification keywords (see below) by specific values. The *k* argument is implied by the *i*, *x* or *e* arguments.
- e This argument indicates that this *get* is for the purpose of making a delta with a later execution of *delta*. It causes creation, or updating of a *p-file* (see FILES). Another *get* with an *e* argument, if at the same delta or for the same new SID, may not be executed until the delta is made. If the *g-file* generated by a *get* with an *e* argument is ruined, a new one may be obtained by executing another *get* with a *k* argument instead of an *e*. Note that although the *c* argument may be used in combination with *e*, *delta* will not use it when regenerating the *g-file* for the purpose of determining what changed. When the *e* argument is supplied the protection restrictions determined by the ceiling, the floor, and the list of users authorized to make deltas are enforced.
- l This argument causes a delta summary to be written into an *l-file* (see FILES). If *-lp* is used then an *l-file* is not created; the delta summary is written on the standard output instead. The *reform(I)* command can be used to truncate lines of the *l-file*.
- p This argument causes the generated text to be written to the standard output instead of to a *g-file*. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the *s* argument is supplied, in which case it disappears.
- s This argument suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.
- m This argument causes each generated text line to be preceded by the SID of the delta which inserted that text line. The format is: SID, followed by a horizontal tab, followed by the text line.
- n This argument causes each generated text line to be preceded with the %M% identification keyword (see below). The format is: %M% identification keyword, followed by a horizontal tab, followed by the text line. When both the *m* and *n*



arguments are supplied the format is: %M% identification keyword, followed by a horizontal tab, followed by the m argument format.

- b This argument is used with the e argument to indicate that the new delta should have an SID in a new branch. This argument is allowed only if the b flag exists in the file; see *admin(1)*.
- g The g argument suppresses the actual getting of source. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.
- t The t argument is used to access the most recent (“top”) delta in a given release (i.e., when no r argument is supplied, or an argument of the form rrel is supplied).

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a newline) before it is processed. If the i argument is supplied included deltas are listed following the notation “Included”; if the x argument is supplied excluded deltas are listed following the notation “Excluded”.

Identifying information is inserted into the generated text by replacing *identification keywords* by appropriate values, wherever they occur. The following keywords are available:

<i>Keyword</i>	<i>Value</i>
%M%	Module name; either the value of the m flag in the file (see <i>admin(1)</i> ), or the <i>g-file</i> name—see FILES.
%I%	SCCS identification string (SID) —%R%.%L%.%B%.%S%.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%E%	Date of newest applied delta (YY/MM/DD).
%G%	Date of newest applied delta (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%U%	Time of newest applied delta (HH:MM:SS).
%Y%	The value of the t flag in the file (see <i>admin(1)</i> ).
%F%	File name.
%C%	Current line number. This keyword is intended for identifying messages output by the program such as “this shouldn’t have happened” type errors. It is <i>not</i> intended to be used on every line to provide sequence numbers.
%Z%	The 4 characters @(#) (used to construct strings recognizable by <i>what(1)</i> ).
%W%	A shorthand notation for constructing <i>what(1)</i> strings for UNIX program files. %W% = %Z%%M% <horizontal-tab>%I%
%A%	Another shorthand notation for constructing <i>what(1)</i> strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%

## FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*,

*p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form "s.modulename", the auxiliary files are named by replacing the leading "s" with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the "s.". For example, if the SCCS file name is "s.xyz.c", the auxiliary file names would be "xyz.c", "l.xyz.c", "p.xyz.c", and "z.xyz.c", respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the *p* argument is supplied, or zero lines of text were generated). It is owned by the real user. If the *k* argument is supplied or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* is also created (unless a *p* follows the *-l*) in the current directory, if the *l* argument is supplied; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory. The *l-file* contains a table showing which deltas were applied. The following is printed for each delta in the SCCS file:

- a) Blank if the delta was applied; "\*" otherwise.
- b) Blank if the delta was applied or wasn't applied and ignored; "\*" if the delta wasn't applied and wasn't ignored.
- c) A code indicating a "special" reason why the delta was or was not applied:
  - "I": Included.
  - "X": Excluded.
  - "C": Cut off (by a *c* argument).
- d) Blank.
- e) SCCS identification string (SID).
- f) Tab character.
- g) Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
- h) Blank.
- i) Creator.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an *e* argument along to *delta*. Its contents are used to prevent a subsequent execution of *get* with an *e* argument until *delta* is executed (subject to the conditions described above under the *e* keyletter description). The *p-file* is created in the directory containing the SCCS file (which might, of course, also be the current directory), and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID this delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time of the *get* ( *no*: the cutoff date-time), followed by a blank and the *-i* keyletter argument if it was present, followed by a blank and the *-x* keyletter argument if it was present, followed by a newline. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same gotten SID or the same new SID.

The *z-file* is created in the directory containing the SCCS file for the duration of updating the *p-file*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444. It serves as a *lock-out* mechanism against simultaneous updates. Its contents are (in binary; 2 bytes) the process ID of the command (i.e., *get*) that created it.

**SEE ALSO**

admin(I), delta(I), prt(I), what(I), help(I), sccsfile(V),  
*SCCS/PWB User's Manual* by L. E. Bonanni and A. L. Glasser.

**DIAGNOSTICS**

Use *help(I)* for explanations.

**BUGS**

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the *e* argument is supplied.

**NAME**

`goto` — command transfer

**SYNOPSIS**

`goto` *label*

**DESCRIPTION**

*Goto* is allowed only when the Shell is taking commands from a file. The file is searched from the beginning for a line beginning with `:` followed by one or more spaces followed by the *label*. If such a line is found, the *goto* command returns. Since the read pointer in the command file points to the line after the label, the effect is to cause the Shell to transfer to the labelled line.

`:` is a do-nothing command that is ignored by the Shell and only serves to place a label.

**SEE ALSO**

`sh(1)`

## NAME

graph — draw a graph

## SYNOPSIS

**graph** [ option ] ... | *plotter*

## DESCRIPTION

*Graph* with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. The graph is written on the standard output to be piped to the *plotter* program for a particular device; see *plot(1)*.

If the coordinates of a point are followed by a nonnumeric string, that string is printed as a label beginning at the point. Labels may be surrounded with quotes "...", in which case they may contain blanks or begin with numeric characters; labels never contain newlines.

The following options are recognized, each as a separate argument.

- a** Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number. A second optional argument is the starting point for the automatic abscissa.
- c** Character string given by next argument is default label for each point.
- d** Omit connections between points. (Disconnect.)
- g***n* Grid style:
  - n*=0, no grid
  - n*=1, axes only
  - n*=2, complete grid (default).
- l** Next argument is label for graph.
- s** Save screen, don't erase before plotting.
- x** Next 1 (or 2) arguments are lower (and upper) *x* (abscissa) limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.
- y** Similarly for *y* (ordinate) axis.
- 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.

Points are connected by straight line segments in the order they appear in input. If a specified lower limit exceeds the upper limit, or if the automatic increment is negative, the graph is plotted upside down. Automatic abscissas begin with the lower *x* limit, or with 0 if no limit is specified. Labels are placed so that the center of an initial letter such as + will fall approximately on the plotting point.

## SEE ALSO

*plot(1)*, *spline(1)*

**BUGS**

*Graph* stores all points internally even when limits are explicit, so utterly enormous graphs can fail unnecessarily.

**NAME**

`grep` — search a file for a pattern

**SYNOPSIS**

`grep [ -v ] [ -b ] [ -c ] [ -n ] [ -s ] expression [ file ] ...`

**DESCRIPTION**

*Grep* searches the input files (standard input default) for lines matching the regular expression. Normally, each line found is copied to the standard output. If the `-v` flag is used, all lines but those matching are printed. If the `-c` flag is used, only a count of matching lines is printed. If the `-n` flag is used, each line is preceded by its relative line number in the file. If the `-b` flag is used, each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

The `-s` flag suppresses the error messages that *grep* would otherwise give for non-existent (or unreadable) files.

In all cases the file name is shown if there is more than one input file.

For a complete description of the regular expression, see *ed*(1). Care should be taken when using the characters `$ * [ ^ | ( )` and `\` in the *expression*, as they are also meaningful to the Shell. It is generally necessary to enclose the entire *expression* argument in quotes.

**SEE ALSO**

*ed*(1), *egrep*(1), *fgrep*(1), *rgrep*(1), *sed*(1), *sh*(1)

**BUGS**

Lines are limited to 256 characters; longer lines are truncated.

Unfortunately, *grep* does not recognize all of the regular expression operators that *ed*(1) does.

## NAME

*gsi* - handle special functions of GSI300 terminal

## SYNOPSIS

*gsi* [+12] [-n] [-d*t,l,c*]

## DESCRIPTION

*Gsi* supports special functions, and optimizes the use, of the GSI300 (DASI300 or DTC300) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time (5 to 70%). *Gsi* can be used to print equations neatly, in the sequence:

```
neqn file ... | nroff | gsi
```

WARNING: if your terminal has a PLOT switch, make sure it is turned ON before *gsi* is used.

The behavior of *gsi* can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

- +12** permits use of 12-pitch, 6 lines/inch text. GSI terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.
- n** controls the size of half-line spacing. A half-line is by default equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of *n* overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, *nroff(1)* half-lines could be made to act as quarter-lines by using **-2**. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **-3** alone, having set the PITCH switch to 12-pitch.
- d*t,l,c*** controls delay factors. The default setting is **-d3,90,30**. GSI terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) requests 2 null bytes per tab (character). The former may be needed for C programs, the latter for files like *lct/passwd*. Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output. The **-d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file *lct/passwd* may be printed using **-d3,30,5**. The value **-d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the carriage return and line feed delays being used at the time: see *GSI300(VII)*. The *sty(1)* modes **n10 cr2** or **n10 cr3** are recommended for most uses.

NOTE: *gsi* always synchronizes its buffering so that it can be used with the *nroff* **-s** flag or **.rd** requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the RETURN key in these cases, you must use the LINE FEED key to get any response.



In many cases, the following sequences are equivalent:

```
nroff -T300 files ...      and  nroff files ... | gsi
nroff -T300-12 files ...   and  nroff files ... | gsi +12
```

The use of *gsi* can thus often be avoided unless special delays or options are required.

Here are the *neqn(I)* names and resulting output for the special characters supported:

Name	Symbol	Name	Symbol
alpha	$\alpha$	OMEGA	$\Omega$
beta	$\beta$	partial	$\partial$
delta	$\delta$	phi	$\phi$
DELTA	$\Delta$	PHI	$\Phi$
epsilon	$\epsilon$	psi	$\psi$
eta	$\eta$	PSI	$\Psi$
gamma	$\gamma$	pi	$\pi$
GAMMA	$\Gamma$	PI	$\Pi$
infinity	$\infty$	rho	$\rho$
integral	$\int$	sigma	$\sigma$
lambda	$\lambda$	SIGMA	$\Sigma$
LAMBDA	$\Lambda$	tau	$\tau$
mu	$\mu$	theta	$\theta$
nabla(del)	$\nabla$	THETA	$\Theta$
not	$\neg$	xi	$\xi$
nu	$\nu$	zeta	$\zeta$
omega	$\omega$		

#### SEE ALSO

450(I), graph(I), greek(V), GSI300(VII), mesg(I), neqn(I), plot(I), stty(I), tabs(I)

#### BUGS

Some characters in the above table can't be correctly printed in column 1 because the print head cannot be moved to the left from there. If your output contains much Greek and/or reverse line feeds, use friction feed instead of a forms tractor. Although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters, and misaligning the first line after a long set of reverse line feeds.

*Gsi* is definitely *not* usable with the "second generation" models of the GSI300, such as the GSI300S or DASI450.

**NAME**

help — ask for help

**SYNOPSIS**

**help** [arg] ...

**DESCRIPTION**

*Help* finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

- type 1     Begins with non-numeric, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., “ge6”, for message 6 from the *get* command).
- type 2     Does not contain numerics (as a command, such as *get* )
- type 3     Is all numeric (e.g., “212”)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try “help stuck”.

**FILES**

The ASCII file searched for the explanatory information for each type of argument is as follows:

- type 1     /usr/lib/help/*prefix-of-argument*
- type 2     /usr/lib/help/cmds
- type 3     /usr/lib/sccs.hf

If the file to be searched for either a type 1 or a type 2 argument does not exist, the search will be attempted on the file for the type 3 argument. In no case, however, will more than one file be searched per argument.

Anyone wishing to modify the files should list out portions of them — the format will be obvious.

**DIAGNOSTICS**

Use *help* for help.

## NAME

`hp` - handle special functions of HP 2640 terminal

## SYNOPSIS

`hp [-e] [-m]`

## DESCRIPTION

*Hp* supports special functions of the Hewlett-Packard 2640 family of terminals, with the primary purpose of producing accurate representations of most *nroff*(I) output. Typical uses are:

`nroff -h files ... | hp`                      or:                      `nroff -h -s files ... | hp`

In the latter case, *nroff* will stop at the beginning of each page including the first and wait for you to hit LINE FEED to initiate output. Regardless of the hardware options on your terminal, *hp* does sensible things with underlining and reverse line feeds. If the terminal has the display enhancements feature, subscripts and superscripts can be indicated in distinct ways. If it has the mathematical-symbol option, you can see Greek and other special characters.

The flags are as follows:

- e it is assumed that your terminal has the display enhancements feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-Bright mode, and subscripts in Half-Bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the display enhancements feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark on light, rather than the usual light on dark.
- m requests minimization of output by removal of newlines. Any contiguous sequence of 3 or more newlines is converted into a sequence of only 2 newlines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other scientific characters, *hp* provides the same set as does *gsi*(I), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*(I).

## DIAGNOSTICS

"line too long" if representation of a line exceeds 300 characters, which would occur, for instance, if you underlined every other character in an 80-character line containing many Greek characters.

## EXIT CODES

- 0 - normal
- 1 - for any error

## SEE ALSO

*gsi*(I), *HP2640*(VII), *neqn*(I), *nroff*(I)

## BUGS

Note that the second or later characters in an overstriking sequence are always assumed to be underlines. For example, a bullet made from lower-case "o" overstruck with "+" appears as an "o" that is either underlined or shown in Inverse Video. Although some terminals do provide numerical superscript characters, no effort is made to display them. The programming is ugly, and most terminals do not possess this feature.

## NAME

if – conditional command

## SYNOPSIS

```
if expr command [ arg ... ]
```

```
if expr then
    command(s)
```

```
    ...
[ else [ command ]
    ... ]
```

```
endif
```

```
test expr
```

## DESCRIPTION

*If* evaluates the expression *expr*. In the first form above, if *expr* is true, the given *command* is executed with the given arguments. The command may be another *if*.

In the second form, if *expr* is true, the commands between the *then* and the next unmatched *else* or *endif* are executed. If *expr* is false, the commands after *then* are skipped, and the commands after the optional *else* are executed. Zero or one commands may be written on the same line as the *else*. In particular, *if* may be used this way. The pseudo commands *else* and *endif* (whichever occurs first) must not be hidden behind semicolons or other commands. This form may be nested: every *then* needs a matching *endif*.

*Test* is an entry to *if* that evaluates the expression and returns exit code 0 if it is true, and code 1 if it is false or in error.

The following primitives are used to construct the *expr*:

**-r file** true if the file exists and is readable.

**-w file** true if the file exists and is writable.

**-s file** true if the file exists and has a size greater than zero.

**-f file** true if the file exists and is an ordinary file.

**-d file** true if the file exists and is a directory.

**-z s1** true if the length of string *s1* is zero.

**-n s1** true if the length of string *s1* is nonzero.

**s1 = s2** true if the strings *s1* and *s2* are equal.

**s1 != s2** true if the strings *s1* and *s2* are not equal.

**n1 -eq n2**

**n1 -ne n2**

**n1 -gt n2**

**n1 -ge n2**

**n1 -lt n2**

**n1 -le n2** true if the stated algebraic relationship exists. The arguments *n1* and *n2* must be integers.

**{ command }** The bracketed command is executed to obtain the exit status. Status zero is considered *true*. The command must **not** be another *if*.

These primaries may be combined with the following operators:

- ! unary negation operator
- a binary *and* operator
- o binary *or* operator
- ( expr ) parentheses for grouping.

-a has higher precedence than -o. Notice that all the operators and flags are separate arguments to *if* and hence must be surrounded by spaces. Notice also that parentheses are meaningful to the Shell and must be escaped.

#### EXIT CODES

- 0 - true expression, no error.
- 1 - false condition or error.

#### SEE ALSO

exit(I), goto(I), sh(I), switch(I), while(I), exit(II)

#### DIAGNOSTICS

- if:missing endif
- if:syntax error: value
- if:non-numeric arg: value
- if:no command: name
- else:missing endif

*Test* may issue any of the *if* messages above, except the first.

#### BUGS

In general, *if*, *else*, *endif*, and *test* must not be hidden behind semicolons on a command line. Many of the effects are obtained by searching the input file and adjusting the read pointer appropriately. Thus, including any of these commands in a part of the file intended to be read by a command other than the shell may cause strange results if they are encountered while searching. These commands ignore redirection or piping of their standard input or output. Commands executed by *if* or *test* may be affected by redirections, but this practice is undesirable and should be avoided.

**NAME**

kill — terminate a process

**SYNOPSIS**

kill [ -signo ] processid ...

**DESCRIPTION**

*Kill* sends signal 15 (terminate) to the specified processes. This will normally terminate the process, unless it is caught. The process number of each asynchronous process started with '&' is reported by the Shell. Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

If a signal number preceded by “-” is given as the first argument, that signal is sent instead. For example, -9 will guarantee a kill.

**SEE ALSO**

*ps*(1), *sh*(1), *kill*(2), *signal*(2)

## NAME

`ld` — link editor

## SYNOPSIS

`ld [ -sulxrdni ] [ -o name ] file ...`

## DESCRIPTION

*Ld* combines several object programs into one; resolves external references; and searches libraries. In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the `-r` option must be given to preserve the relocation bits.) The output of *ld* is left on **a.out**. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. The entry point of the output is the function named **main**.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

*Ld* understands several flag arguments which are written preceded by a '-'. Except for `-l`, they should appear before the file names.

- `-s` 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1).
- `-u` Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.
- `-l` This option is an abbreviation for a library name. `-l` alone stands for `'/lib/liba.a'`, which is the standard system library for assembly language programs. `-lx` stands for `'/lib/libx.a'`, where *x* can be a string. If that does not exist, *ld* tries `'/usr/lib/libx.a'`. A library is searched when its name is encountered, so the placement of a `-l` is significant.
- `-x` Do not preserve local (non-globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.
- `-X` Save local symbols except for those whose names begin with 'L'. This option is used by *cc* to discard internally generated labels while retaining symbols local to routines.
- `-r` Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.
- `-d` Force definition of common storage even if the `-r` flag is present.
- `-n` Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text.
- `-i` When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and `-n` is that here the data starts at location 0.

**-o** The *name* argument after **-o** is used as the name of the *ld* output file, instead of **a.out**.

**FILES**

/lib/lib?.a	libraries
/usr/lib/lib?.a	more libraries
a.out	output file

**SEE ALSO**

as(1), ar(1), cc(1)



## NAME

lex — generate programs for simple lexical tasks

## SYNOPSIS

lex [ -[rctvfn] ] [file] ...

## DESCRIPTION

*Lex* generates programs to be used in simple lexical analysis of text.

The input file(s) contain strings and expressions to be searched for, and C text to be executed when found. A file "lex.yy.c" is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in

```
[abx-z]
```

to indicate a, b, x, y, and z; and the operators \*, +, and ? mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character '.' is the class of all ASCII characters except newline. Parentheses for grouping and vertical bar for alternation are also supported. The character ^ at the beginning of an expression permits a successful match only immediately after a newline, and the character % at the end of an expression requires a trailing newline. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \. Thus

```
[a-zA-Z]+
```

matches a string of letters.

Three subroutines defined as macros are expected: *input()* to read a character; *unput(c)* to replace a character read; and *output(c)* to place an output character. They are defined in terms of the standard streams (and the -IS standard I/O library), but you can override them. The program generated is named *yylex()*, and the library contains a *main()* which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function *yymore()* accumulates additional characters into the same *yytext*; and the function *yyless(p)* pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yytext*. The macros *input* and *output* use files "yyin" and "yyout" to read from and write to, defaulted to "stdin" and "stdout", respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the "lex.yy.c" file. All rules should follow a %% as in YACC. Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with {}. Note that curly brackets do not imply parentheses; only string substitution is done. Example:

```
D      [0-9]
%%
if      printf("IF statement\n");
[a-z]+ printf("tag, value %s\n",yytext);
0{D}+  printf("octal number %s\n",yytext);
{D}+   printf("decimal number %s\n",yytext);
"++"   printf("unary op\n");
"+"    printf("binary op\n");
"/*"   {      loop:
          while (input() != '*');
```

```

switch (input())
{
    case '/': break;
    case '*': unput('*');
    default: go to loop;
}
}

```

The external names generated by *lex* all begin with the prefix "yy" or "YY".

The flags must appear before any files. The flag `-r` indicates Ratfor actions, `-c` indicates C actions and is the default, `-t` causes the "lex.yy.c" program to be written instead to standard output, `-v` provides a one-line summary of statistics of the machine generated, `-f` indicates "faster" compilation, so no packing is done, but it can handle much smaller machines only, `-n` will not print out the `-` summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

This is intended to replace the older version of Lex. The new standard I/O library is used, so actions must use it, and an "include" statement is automatically provided. A definition in the definitions section may refer to other definitions (but not to itself). The "%+" option has been eliminated. The notation `r{d,e}` in a rule indicates between `d` and `e` instances of regular expression `r`. It has higher precedence than `|`, but lower than `*`, `?`, `+`, and concatenation.

In the definitions section,

```

%p num  sets the max. — of positions to num (dft = 2000)
%n num  sets the max. — of states to num (dft = 500)
%t num  sets the max. — of parse tree nodes to num (dft = 1000)
%a num  sets the max. — of transitions to num (dft = 3000)

```

The use of one or more of the above automatically implies the `-v` option, unless the `-n` option is used.

#### SEE ALSO

yacc(1)

*LEX* — *Lexical Analyzer Generator* by M. E. Lesk and E. Schmidt.

#### BUGS

The Ratfor option is not yet fully operational.

**NAME**

*ln* - make a link

**SYNOPSIS**

*ln* name1 [ name2 ]

**DESCRIPTION**

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There is no way to distinguish a link to a file from its original directory entry; any changes in the file are effective independently of the name by which the file is known.

*Ln* creates a link to an existing file *name1*. If *name2* is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of *name1*.

It is forbidden to link to a directory or to link across file systems.

**SEE ALSO**

*rm*(1)

**BUGS**

There is nothing particularly wrong with *ln*, but *tp* doesn't understand about links and makes one copy for each name by which a file is known; thus if the tape is extracted several copies are restored and the information that links were involved is lost.

**NAME**

login — sign onto UNIX

**SYNOPSIS**

**login** [ username ]

**DESCRIPTION**

The *login* command is used when a user initially signs onto UNIX, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See 'How to Get Started' for how to dial up initially.

If *login* is invoked without an argument, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of *.mail* and message-of-the-day files. *Login* initializes the user and group IDs and the working directory, then executes a command interpreter (usually *sh*(1)) according to specifications found in a password file.

Login is recognized by the Shell and executed directly (without forking).

**FILES**

/etc/utmp	accounting
/usr/adm/wtmp	accounting
.mail	mail
/etc/motd	message-of-the-day
/etc/passwd	password file

**SEE ALSO**

init(VIII), getty(VIII), mail(I), passwd(I), passwd(V), sh(I), su(I)

**DIAGNOSTICS**

'login incorrect,' if the name or the password is bad. 'No Shell', 'cannot open password file,' 'no directory': consult a UNIX programming counselor. System hangs up a line left in login state.

**NAME**

logname, logdir, logtty — information from login

**SYNOPSIS**

**logname**

**logdir**

**logtty**

**DESCRIPTION**

*Logname* prints the user's login name.

*Logdir* prints the login directory pathname.

*Logtty* prints the single character tty letter (and never prints 'x').

These data are created by *login*(1).

## NAME

ls — list contents of directory

## SYNOPSIS

ls [ **-l** **-t** **-a** **-d** **-r** **-u** **-i** **-f** **-g** ] name ...

## DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. There are several options:

- l** list in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers.
- t** sort by time modified (latest first) instead of by name, as is normal
- a** list all entries; usually those beginning with '.' are suppressed
- s** give size in blocks for each entry
- d** if argument is a directory, list only its name, not its contents (mostly used with **-l** to get status on directory)
- r** reverse the order of sort to get reverse alphabetic or oldest first as appropriate
- u** use time of last access instead of last modification for sorting (**-t**) or printing (**-l**)
- i** print i-number in first column of the report for each file listed
- f** force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.
- g** Give group ID instead of owner ID in long listing.

The mode printed under the **-l** option contains 11 characters which are interpreted as follows: the first character is

- d** if the entry is a directory;
- b** if the entry is a block-type special file;
- c** if the entry is a character-type special file;
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

- r** if the file is readable
- w** if the file is writable
- x** if the file is executable
- if the indicated permission is not granted

The group-execute permission character is given as `s` if the file has set-group-ID mode; likewise the user-execute permission character is given as `S` if the file has set-user-ID mode.

The last character of the mode is normally blank but is printed as `t` if the 1000 bit of the mode is on. See *chmod(1)* for the current meaning of this mode.

**FILES**

`/etc/passwd` to get user ID's for `ls -l`.

## NAME

m4 — macro processor

## SYNOPSIS

m4 [ files ]

## DESCRIPTION

*M4* is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no arguments, or if an argument is '-', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

```
name(arg1,arg2,..., argn)
```

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '\_', where the first character is not a digit.

Left and right single quotes (") are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

*M4* makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

- define** The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of \$*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string.
- undefine** removes the definition of the macro named in its argument.
- ifdef** If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null.
- changequote** Change quote characters to the first and second arguments. *Changequote* without arguments restores the original values (i.e., ").
- divert** *M4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
- undivert** causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
- divnum** returns the value of the current output stream.
- dnl** reads and discards characters up to and including the next newline.



- ifelse** has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
- incr** returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
- eval** evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, \*, /, %, ^ (exponentiation); relationals; parentheses.
- len** returns the number of characters in its argument.
- index** returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.
- substr** returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
- include** returns the contents of the file named in the argument.
- sinclude** is identical to *include*, except that it says nothing if the file is inaccessible.
- syscmd** executes the UNIX command given in the first argument.
- errprint** prints its argument on the diagnostic output file.
- dumpdef** prints current names and definitions, for the named items, or for all if no arguments are given.

**SEE ALSO**

*The M4 Macro Processor* by B. W. Kernighan and D. M. Ritchie.

## NAME

mail — send mail to designated users

## SYNOPSIS

mail [-yn] [ person ... ]  
mail -f file

## DESCRIPTION

*Mail* with no argument searches for a file called **.mail**, prints it in reverse chronological order if it is nonempty, then asks if it should be saved. If the answer is **y**, the mail is added to **mbox**. In either case, **.mail** is truncated to zero length. To leave **.mail** untouched, hit 'delete.' The question can be answered on the command line with the argument **-y** or **-n**.

*Mail* tries to use **.mail** and **mbox** in the current directory. But if **.mail** doesn't exist, *mail* uses **.mail** and **mbox** in your *login* directory instead.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or a line with just '.') and adds it to each *person's* **.mail** file. The message is preceded by the sender's name and a post-mark. A *person* is a user name recognized by *login*(1). Mail is sent to the *login* directory of that user.

The **-f** option causes the named file to be printed as if it were mail.

When a user logs in he is informed of the presence of mail.

To receive mail, a **.mail** file must exist in your *login* directory, and it must be writable by everyone. However, it need not be readable by everyone.

## FILES

/etc/passwd	to identify sender and locate persons
.mail	input mail
mbox	saved mail
/tmp/m????	temp file

## SEE ALSO

write(1)

**NAME**

make — make a program

**SYNOPSIS**

**make** [**-f** *descfile*] [**-p**] [**-i**] [**-s**] [**-r**] [**-n**] [**-t**] *file* ...

**DESCRIPTION**

*Make* may be used to mechanize program creation and maintenance, while ensuring that all constituents are current. A graph of dependencies is specified in the *descfile(s)*. The standard input will be read if **-** is given for *descfile*. If no **-f** options are present, the file named **makefile** or, if absent, the file named **Makefile** in the current directory is used. The **-p** option prints out a version of that graph. Each file name argument is 'made', as described below. If no such arguments are present, the initial node in the description file is made.

The *descfile* consists of a sequence of entries that describe the prerequisites and operations for creating an object (usually a file). The first line of each entry contains the names of the objects to be made, followed by a colon, optionally followed by a list of other files that must be available and current in order to remake it. Text following a semicolon on the first line, and all immediately following lines that begin with a tab, are fed to the Shell to make the object. Each line is fed to a separate instance of the Shell. All text following a sharp is taken to be a comment. For example:

```
pgm: x.o y.o ; cc x.o y.o -lp
      mv a.out pgm    # command to be done
x.o: dcls
```

*Make* walks the graph of dependencies. If a needed object depends on another that is not present or is younger than itself, it is remade. If an object's name ends in '.o', the description file, and then the current directory, are searched for a corresponding name ending in '.r', '.f', '.c', '.s', '.l' (Lex), '.y' (Yacc-C). (These default rules are not applied if the **-r** option is specified). If such a file is found and is younger than the object, a compilation command is executed. In the example above, if 'dcls' has been changed since 'pgm' was last made, 'x.c' will be recompiled and 'pgm' will be reloaded.

Each command line is printed before it is executed unless the **-s** option is specified on the command line or the special name '.SILENT' appears in the description. The command lines are printed but not executed if the **-n** option is specified. The date last modified is updated but the commands given are not executed for each file if the **-t** option is specified. (This option is useful when a source change is known to be incremental or benign).

*Make* examines the *exit*(II) status returned by each executed command. If the status is non-zero (i.e., if an error occurred), *make* aborts, unless either (a) the **-i** option was specified, or (b) the command name in the *descfile* was prefixed by '-'.

**SEE ALSO**

*Make — A Program for Maintaining Computer Programs* by S. I. Feldman.

**DIAGNOSTICS**

No description file argument  
 Cannot find description file  
 Syntax error  
 Don't know how to make xxx.

**BUGS**

Many UNIX commands return random status, which will cause *make* to assume that the command failed. In case of trouble, use the `-i` option or a minus sign on the command line.

## NAME

`man` – print on-line documentation

## SYNOPSIS

`man` [ options ] documents ...

## DESCRIPTION

*Man* is a Shell command file that prints on-line documentation on the standard output by means of *nroff*(1) or *troff*(1). On-line documentation consists of manual pages from the PWB/UNIX User's Manual.

The command line to print manual pages consists of:

`man` [ term ] [ -s ] [ section ] title ...

where "term" is one of the following:

- t produces photocomposed output;
- g adapts the output to a DASI300 terminal in 12-pitch mode;
- 450 adapts the output to the DASI450 terminal in 10-pitch mode;
- hp adapts the output to a Hewlett-Packard terminal;
- v followed by a space and a bin number, produces output on the Versatec printer. Exactly one bin number *must* be specified when the -v option is used.

The -s option prints only the SYNOPSIS portion of a manual page.

*Section* is the section number in the PWB/UNIX User's Manual in which a manual page is filed; it is specified as a single Arabic decimal digit. If *section* is omitted on the command line, the section number defaults to 1. If the page is not in the given section, then a search is made of *all* sections of the manual. If the page is not found (i.e., does not exist), an error message is produced.

*Title* is the name of the manual page. One or more titles may be specified in a single command.

Thus, the command line:

`man -g 1 ed man tbl`

would print out (in 12-pitch on a DASI300 terminal) the pages for the commands *ed*, *man*, and *tbl*, all of which can be found in Section I of the PWB/UNIX User's Manual.

## DIAGNOSTICS

"man page for xxx not found"      A manual page for xxx does not exist.

## FILES

/usr/man/man[0-8]      Installed PWB/UNIX pages.

## SEE ALSO

Anyone who wishes to *write* manual pages like those accessed by this command should read *PWB/UNIX Manual Page Macros* by E. M. Piskorik.

**NAME**

**msg** — permit or deny messages

**SYNOPSIS**

**msg** [ **n** ] [ **y** ]

**DESCRIPTION**

*Msg* with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Msg* with argument **y** reinstates permission. All by itself, *msg* reverses the current permission. In all cases the previous state is reported.

**FILES**

/dev/tty?

**SEE ALSO**

*write*(1)

**DIAGNOSTICS**

'?' if the standard input file is not a terminal.

**NAME**

**mkdir** — make a directory

**SYNOPSIS**

**mkdir** dirname ...

**DESCRIPTION**

*Mkdir* creates specified directories in mode 755. The standard entries '.' and '..' are made automatically.

**SEE ALSO**

**rmdir**(1)

## NAME

`mm` – run off document with PWB/MM

## SYNOPSIS

`mm` [options] [files]

## DESCRIPTION

The `mm` command can be used to run off documents using `nroff`(I) and the PWB/MM text formatting codes. It has options to specify preprocessing by `tbl`(I) or by `neqn`(I) and for postprocessing by various output filters. The proper pipe sequences and the required arguments and flags for `nroff`(I) and PWB/MM are generated, depending on the options selected. For example, inclusion of the `-h nroff`(I) flag occurs unless `col`(I) is to be used or unless the `-450` option is specified.

The options for `mm` are listed below. Any other arguments or flags, e.g. `-rC3`, are passed to `nroff`(I) or to PWB/MM, as appropriate. The options can occur in any order, but they must appear before the files.

- `-e` `neqn`(I) processing is needed.
- `-t` `tbl`(I) is needed.
- `-c` `col`(I) is needed.
- `-12` want 12 pitch mode. (Be sure that the 12-pitch switch is set on the terminal.)
- `-300` output onto a DASI 300 terminal. This is the *default* terminal type.
- `-hp` output onto a HP 2640A.
- `-450` output onto a DASI 450.
- `-300S` output onto a DASI 300S.
- `-300s` output onto a DASI 300S.
- `-tn` output onto a GE TermiNet 300.
- `-tn300` output onto a GE TermiNet 300.
- `-ti` output onto a Texas Instrument terminal.
- `-37` output onto a TTY 37.

If several terminal types are specified, the last one takes precedence. Note that `-ti`, `-tn`, and `-tn300` all do the same thing; they all imply `-c`, and work for any terminal that lacks reverse line feed capability.

As an example,

```
mm -t -450 -rC3 -12 qqsv*
```

generates

```
tbl qqsv* | nroff -h -mm -rT1 -rC3 - | 450
```

If no arguments are given, `mm` prints a list of options.

If only options and unreadable files are specified, then `mm` terminates silently.

## HINTS

1. `mm` may invoke `nroff`(I) with the `-h` flag. With this flag, `nroff`(I) assumes that the terminal has tabs set at every 8 character positions.



2. Use the `-olist` option of *nroff*(1) to specify ranges of pages to be output.
3. When either `-t` or `-e` or both are specified and the `-olist` does *not* cause the last page of the document to be printed, a "broken pipe" message from the Shell will result.

**SEE ALSO***nroff*(1)*PWBMM — Programmer's Workbench Memorandum Macros* by D. W. Smith and J. R. Mashey.*Typing Documents with PWBMM* by D. W. Smith and E. M. Piskorik.

**NAME**

**mv** — move or rename a file

**SYNOPSIS**

**mv** name1 name2

**DESCRIPTION**

*Mv* changes the name of *name1* to *name2*. If *name2* is a directory, *name1* is moved to that directory with its original file-name. Directories may only be moved within the same parent directory (just renamed).

If *name2* already exists, it is removed before *name1* is renamed. If *name2* has a mode which forbids writing, *mv* prints the mode and reads the standard input to obtain a line; if the line begins with *y*, the move takes place; if not, *mv* exits.

If *name2* would lie on a different file system, so that a simple rename is impossible, *mv* copies the file and deletes the original.

**SEE ALSO**

**cp(I)**, **cpio(I)**

**BUGS**

It should take a **-f** flag, like *rm*, to suppress the question if the target exists and is not writable.

**NAME**

`neqn` — typeset mathematics on terminal

**SYNOPSIS**

`neqn` [ file ] ...

**DESCRIPTION**

*Neqn* is an *nroff*(1) preprocessor. The input language is the same as that of *eqn*(1). Normal usage is almost always:

```
neqn file ... | nroff
```

Output is meant for terminals with forward and reverse capabilities, such as the Model 37 TELETYPE® or GSI (DASI or DIABLO) terminals.

If no arguments are specified, *neqn* reads the standard input, so it may be used as a filter.

**SEE ALSO**

*eqn*(1), *gsi*(1), *mm*(1), *DASI450*(VII), *GSI300*(VII)

*Typesetting Mathematics — User's Guide (2nd Edition)* by B. W. Kernighan and L. L. Cherry.

*New Graphic Symbols for EQN and NEQN* by C. Scrocca.

**BUGS**

Because of some interactions with *nroff*(1), there may not always be enough space left before and after lines containing equations.

**NAME**

`newgrp` — log in to a new group

**SYNOPSIS**

`newgrp` group

**DESCRIPTION**

*Newgrp* changes the group identification of its caller, analogously to *login*(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

A password is demanded if the group has a password and the user himself does not.

When most users log in, they are members of the group named 'other.'

**FILES**

/etc/group, /etc/passwd

**SEE ALSO**

*login*(1), *group*(V)

**NAME**

`next` – new standard input for shell procedure

**SYNOPSIS**

`next` [ *filename* ]

**DESCRIPTION**

This command causes *filename* to become standard input. The current input is never resumed. If no filename is given, the real terminal is assumed.

*Next* is executed within the shell.

**SEE ALSO**

`sh(1)`

**NAME**

nice — run a command at low priority

**SYNOPSIS**

nice [ *-number* ] *command* [ *arguments* ]

**DESCRIPTION**

*Nice* executes *command* with low scheduling priority. If the *number* argument is given, that priority (in the range 1–20) is used; if not, priority 4 is used.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. ‘--10’.

**SEE ALSO**

nohup(I), nice(II)

**NAME**

**nm** — print name list

**SYNOPSIS**

**nm** [ **-cnrupg** ] [ name ]

**DESCRIPTION**

*Nm* prints the symbol table from the output file of a compiler or loader run. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined) **A** (absolute) **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

If no file is given, the symbols in **a.out** are listed.

Options are:

- c** list only C-style external symbols, that is those beginning with underscore ‘\_’.
- g** print only global (external) symbols
- n** sort by value instead of by name
- p** don’t sort; print in symbol-table order
- r** sort in reverse order
- u** print only undefined symbols.

**FILES**

a.out

**NAME**

**nohup** – run a command immune to hangups

**SYNOPSIS**

**nohup** *command* [ *arguments* ]

**DESCRIPTION**

*Nohup* executes *command* with hangups, quits and interrupts all ignored. If output is not re-directed by the user, it will be sent to */dev/null* (a “write-only” file).

**SEE ALSO**

*nice*(1), *signal*(2)



## NAME

nroff, troff — text formatters

## SYNOPSIS

**nroff** (or **troff**) [ options ] files

## DESCRIPTION

NROFF and TROFF accept lines of text interspersed with lines of format control information and format the text into a printable, paginated document having a user-designed style. NROFF and TROFF are highly compatible with each other and it is almost always possible to prepare input acceptable to both. Conditional input is provided that enables the user to embed input expressly destined for either program. NROFF can prepare output directly for a variety of terminal types and is capable of utilizing the full resolution of each terminal.

An argument consisting of a single minus (—) is taken to be a file name corresponding to the standard input. If no file names are given, input is taken from the standard input. The options, which may appear in any order so long as they appear before the filenames, are:

<i>Option</i>	<i>Effect</i>
<b>-olist</b>	Print only pages whose page numbers appear in <i>list</i> , which consists of numbers and number ranges separated by commas. A number range has the form <i>N-M</i> and means pages <i>N</i> through <i>M</i> inclusive; an initial <i>-N</i> means from the beginning to page <i>N</i> ; and a final <i>N-</i> means from <i>N</i> to the end.
<b>-nN</b>	Number first generated page <i>N</i> .
<b>-sN</b>	Stop every <i>N</i> pages. NROFF will halt prior to every <i>N</i> pages (default <i>N=1</i> ) to allow paper loading or changing, and will resume upon receipt of a new-line character. TROFF will stop the phototypesetter every <i>N</i> pages, produce a trailer to allow the changing of cassettes, and will resume after the phototypesetter START button is pressed.
<b>-mname</b>	Prepends the macro file <i>/usr/lib/tmac.name</i> to the input files.
<b>-raN</b>	Register <i>a</i> (one-character name) is set to <i>N</i> .
<b>-i</b>	Read standard input after the input files are exhausted.
<b>-q</b>	Invoke the simultaneous input-output mode of the <b>rd</b> request.

*NROFF Only*

<b>-Ttype</b>	Specifies the output terminal type. Currently defined values for <i>type</i> are <b>37</b> for the (default) Model 37 TELETYPE®, <b>tn300</b> for the GE TermiNet 300 (or any terminal without half-line capabilities), <b>300</b> for the DASI-300, <b>450</b> for the DASI-450 (or Diablo Hyterm) and <b>300S</b> for the DASI-300S. For 12-pitch, use <b>300-12</b> , <b>300S-12</b> , and <b>450-12</b> .
<b>-e</b>	Produce equally-spaced words in adjusted lines, using full terminal resolution.
<b>-h</b>	Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

*TROFF Only*

<b>-t</b>	Direct output to the standard output instead of the phototypesetter.
<b>-f</b>	Refrain from feeding out paper and stopping phototypesetter at the end of the run.

- w Wait until phototypesetter is available, if it is currently busy.
- b TROFF will report whether the phototypesetter is busy or available. No text processing is done.
- a Send a printable (ASCII) approximation of the results to the standard output.
- pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
- g Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output.

**FILES**

/usr/lib/suftab suffix hyphenation tables  
/tmp/ta00000 temporary file  
/usr/lib/tmac.\* standard macro files  
/usr/lib/term/\* (NROFF only) terminal driving tables  
/usr/lib/font/\* (TROFF only) font width tables

**SEE ALSO**

*NROFF/TROFF User's Manual* by J. F. Ossanna.  
*A TROFF Tutorial* by B. W. Kernighan.  
tbl(1).  
For NROFF, see neqn(1), col(1), and tabs(1)  
For TROFF, see eqn(1).

## NAME

`od` - octal dump

## SYNOPSIS

`od [ -abcdho ] [ file ] [ [ + ] offset[ . ] [ b ] ]`

## DESCRIPTION

*Od* dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, `-o` is default. The meanings of the format argument characters are:

- a** interprets words as PDP-11 instructions and dis-assembles the operation code. Unknown operation codes print as ???.
- b** interprets bytes in octal.
- c** interprets bytes in ascii. Unknown ascii characters are printed as \?.
- d** interprets words in decimal.
- h** interprets words in hex.
- o** interprets words in octal.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used. Thus *od* can be used as a filter.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If `.` is appended, the offset is interpreted in decimal. If `b` is appended, the offset is interpreted in blocks. (A block is 512 bytes.) If the file argument is omitted, the offset argument must be preceded by `+`.

Dumping continues until end-of-file.

## SEE ALSO

`db(1)`

**NAME**

`onintr` — handle interrupts in shell files

**SYNOPSIS**

`onintr` [ label ]

**DESCRIPTION**

The *onintr* command catches interrupts received while the Shell is reading from a file. After the interrupt, and after any active process has completed, the Shell procedure is transferred to the label specified. Unless another *onintr* command is processed, the next interrupt will kill the Shell. The command without a label turns interrupts back on. The special case “*onintr* —” causes interrupts to be totally ignored, both by the Shell itself and by subsequent commands invoked by the Shell.

*Onintr* is executed within the Shell.

**SEE ALSO**

`sh(1)`

**NAME**

passwd — change login password

**SYNOPSIS**

**passwd** name password

**DESCRIPTION**

The *password* becomes associated with the given login *name*. This can only be done by the user who has that login name, or by the super-user. An explicit null argument ("") for the *password* argument removes any password.

**FILES**

/etc/passwd

**SEE ALSO**

login(I), passwd(V), crypt(III)

**NAME**

plot: t300, t300s, t450 – graphics filters

**SYNOPSIS**

**t300**

**t300s**

**t450**

**DESCRIPTION**

These commands read plotting instructions (see *plot(V)*) from the standard input, and produce device-dependent plotting instructions on the standard output.

*T300* produces a plot for a GSI 300 terminal on the standard output.

*T300s* produces a plot for a GSI 300s terminal on the standard output.

*T450* produces a plot for a DASI 450 terminal on the standard output.

**SEE ALSO**

graph(I), plot(III), plot(V)

**NAME**

`pr` - print file

**SYNOPSIS**

`pr [ -h header ] [ -n ] [ +n ] [ -wn ] [ -ln ] [ -t ] [ -sc ] [ -m ] name ...`

**DESCRIPTION**

*Pr* produces a printed listing of one or more files. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, *pr* prints its standard input, and is thus usable as a filter.

Options apply to all following files but may be reset between files:

- n produce *n*-column output
- +n begin printing with page *n*
- h treat the next argument as a header to be used instead of the file name
- wn for purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72
- ln take the length of the page to be *n* lines instead of the default 66
- t do not print the 5-line header or the 5-line trailer normally supplied for each page
- sc separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.
- m print all files simultaneously, each in one column

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

**FILES**

`/dev/tty?` to suspend messages.

**SEE ALSO**

`cat`(1), `cp`(1)

**DIAGNOSTICS**

none; files not found are ignored.

**NAME**

prof - display profile data

**SYNOPSIS**

**prof** [ **-a** ] [ **-l** ] [ file ]

**DESCRIPTION**

*Prof* interprets the file *mon.out* produced by the *monitor*(III) subroutine. Under default modes, the symbol table in the named object *file* (**a.out** default) is read and correlated with the **mon.out** profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the **-a** option is used, all symbols are reported rather than just external symbols. If the **-l** option is used, the output is listed by symbol value rather than decreasing percentage.

In order for the number of calls to a routine to be tallied, the **-p** option of *cc* must have been given when the file containing the routine was compiled. This option also arranges for the **mon.out** file to be produced automatically.

**FILES**

mon.out	for profile
a.out	for namelist

**SEE ALSO**

*monitor*(III), *profil*(II), *cc*(I)

**BUGS**

Beware of quantization errors.



## NAME

`prt` - print SCCS file

## SYNOPSIS

```
prt [-d] [-s] [-a] [-i] [-u] [-f] [-t] [-b] [-e] [-y[SID]]
[-c[cutoff]] [-r[reverse-cutoff]] name ...
```

## DESCRIPTION

*Prt* prints part or all of an SCCS file in a useful format. If a directory is named, *prt* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with "s."), and unreadable files are silently ignored. If a name of "-" is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

- d This keyletter normally causes the printing of delta table entries of the "D" type.
- s Causes only the first line of the delta table entries to be printed; that is, only up to the statistics. This keyletter is effective only if the **d** keyletter is also specified (or assumed).
- a Causes those types of deltas normally not printed by the **d** keyletter to be printed. These are types "R" (removed) and "H" (changed history). This keyletter is effective only if the **d** keyletter is also specified (or assumed).
- i Causes the printing of the serial numbers of those deltas included, excluded, and ignored. This keyletter is effective only if the **d** keyletter is also specified (or assumed).

The following format is used to print those portions of the SCCS file as specified by the above keyletters. The printing of each delta table entry is preceded by a newline character.

- a) Type of delta ("D", "R", or "H").
- b) Space.
- c) SCCS identification string (SID).
- d) Tab.
- e) Date and time of creation.  
(in the form YY/MM/DD HH:MM:SS)
- f) Tab.
- g) Creator.
- h) Tab.
- i) Serial number.
- j) Tab.
- k) Predecessor delta's serial number.
- l) Tab.
- m) Statistics.  
(in the form inserted/deleted/unchanged)
- n) Newline.

- o) "Included:tab", followed by SID's of deltas included, followed by newline (only if there were any such deltas and if i keyletter was supplied).
  - p) "Excluded:tab", followed by SID's of deltas excluded, followed by newline (see note above).
  - q) "Ignored:tab", followed by SID's of deltas ignored, followed by newline (see note above).
  - r) "MRs:tab", followed by MR numbers related to the delta, followed by newline (only if any MR numbers were supplied).
  - s) Lines of comments (history), followed by newline (if any were supplied).
- u Causes the printing of the login-names of those users allowed to make deltas.
  - f Causes the printing of the flags of the named file.
  - t Causes the printing of the descriptive text contained in the file.
  - b Causes the printing of the body of the SCCS file.
  - e This keyletter implies the d, i, u, f, and t keyletters and is provided for convenience.
  - y This keyletter will cause the printing of the delta table entries to stop when the delta just printed has the specified SID. If no delta in the table has the specified SID, the entire table is printed. If no SID is specified, the first delta in the delta table is printed. This keyletter will cause the entire delta table entry for each delta to be printed as a single line (the newlines in the normal multi-line format of the d keyletter are replaced by blanks) preceded by the name of the SCCS file being processed, followed by a ":", followed by a tab. This keyletter is effective only if the d keyletter is also specified (or assumed).
  - c This keyletter will cause the printing of the delta table entries to stop if the delta about to be printed is older than the specified cutoff date-time (see *ger(I)* for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. As with the y keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a ":", followed by a tab. This keyletter is effective only if the d keyletter is also specified (or assumed).
  - r This keyletter will cause the printing of the delta table entries to begin when the delta about to be printed is older than or equal to the specified cutoff date-time (see *ger(I)* for the format of date-time). If no date-time is supplied, the epoch 0000 GMT Jan. 1, 1970 is used. (In this case, nothing will be printed). As with the y keyletter, this keyletter will cause the entire delta table entry to be printed as a single line and to be preceded by the name of the SCCS file being processed, followed by a ":", followed by a tab. This keyletter is effective only if the d keyletter is also specified (or assumed).

If any keyletter but y, c, or r is supplied, the name of the file being processed (preceded by one newline and followed by two newlines) is printed before its contents.

If none of the u, f, t, or b keyletters is supplied, the d keyletter is assumed.

Note that the s and i keyletters, and the c and r keyletters are mutually exclusive; therefore, they may not be specified together on the same *prt* command.

The form of the delta table as produced by the `y`, `c`, and `r` keyletters makes it easy to sort multiple delta tables by time order. For example, the following will print the delta tables of all SCCS files in directory `sccs` in reverse chronological order:

```
prt -c sccs | grep . | sort -rttab +2 -3
```

When both the `y` and `c` or the `y` and `r` keyletters are supplied, `prt` will stop printing when the first of the two conditions is met.

The `reform(I)` command can be used to truncate long lines.

See `admin(I)`, `scsfile(V)`, and *SCCS/PWB User's Manual* for more information about the meaning of the output of `prt`.

#### SEE ALSO

`admin(I)`, `get(I)`, `delta(I)`, `what(I)`, `help(I)`, `scsfile(V)`  
*SCCS/PWB User's Manual* by L. E. Bonanni and A. L. Glasser.

#### DIAGNOSTICS

Use `help(I)` for explanations.

## NAME

ps — process status

## SYNOPSIS

ps [ *aklxt* ] [ *namelist* ]

## DESCRIPTION

*Ps* prints certain indicia about active processes. The *a* flag asks for information about all processes with terminals (ordinarily only one's own processes are displayed); *x* asks even about processes with no terminal; *l* asks for a long listing. The short listing contains the process ID, tty letter, the cumulative execution time of the process and an approximation to the command line. If the *k* flag is specified, the file */sys/sys/core* is used in place of */dev/mem*. This is used for postmortem system debugging. If a second argument is given, it is taken to be the file containing the system's namelist. If the *t* flag is used, the following character is taken to be the specific tty for which information is to be printed.

The long listing is columnar and contains

F	Flags associated with the process. 01: in core; 02: system process; 04: locked in core (e.g. for physical I/O); 10: being swapped; 20: being traced by another process.
S	The state of the process. 0: nonexistent; S: sleeping; W: waiting; R: running; I: intermediate; Z: terminated; T: stopped.
UID	The user ID of the process owner.
PID	The process ID of the process; as in certain cults it is possible to kill a process if you know its true name.
PPID	The process ID of the parent process.
CPU	Processor utilization for scheduling.
PRI	The priority of the process; high numbers mean low priority.
NICE	Used in priority computation.
ADDR	The core address of the process if resident, otherwise the disk address.
SZ	The size in blocks of the core image of the process.
WCHAN	The event for which the process is waiting or sleeping; if blank, the process is running.
TTY	The controlling tty for the process.
TIME	The cumulative execution time for the process.

COMMAND The command and its arguments.

*Ps* makes an educated guess as to the file name and arguments given when the process was created by examining core memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

## FILES

<i>/unix</i>	system namelist
<i>/dev/mem</i>	core memory

/sys/sys/core alternate core file  
/dev searched to find swap device and tty names

SEE ALSO  
kill(I)

**NAME**

`ptx` — permuted index

**SYNOPSIS**

`ptx` [ `-t` ] *input* [ *output* ]

**DESCRIPTION**

*Ptx* generates a permuted index from file *input* on file *output*. It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally the sorted lines are rotated so the keyword comes at the middle of the page.

*Input* should be edited to remove useless lines. The following words are suppressed: 'a', 'an', 'and', 'as', 'is', 'for', 'of', 'on', 'or', 'the', 'to', 'up'.

The optional argument `-t` causes *ptx* to prepare its output for the phototypesetter.

The index for this manual was generated using *ptx*.

**FILES**

/bin/sort

## NAME

**pump** — Shell data transfer command

## SYNOPSIS

**pump** [ **-**[subchar] ] [ **+** ] [ eofstr ]

## DESCRIPTION

*Pump* is a filter that copies its standard input to standard output with possible substitution of Shell arguments and variables. It reads its input to end-of-file, or until it finds *eofstr* alone on a line. If not specified, *eofstr* is assumed to be '!'. Normally, Shell variable and argument values are substituted in the data stream, using '\$' as the character to indicate their presence. The argument '-' alone suppresses all substitution, '-subchar' causes *subchar* to be used as the indicator character for substitution in place of '\$'. Escaping is handled as in double quoted(") strings: the indicator character may be hidden by preceding it with a '\'. Otherwise, '\' and other characters are transmitted unchanged. The '+' flag causes all leading tab characters in the input to be thrown away, in order to permit readable indentation of text and *eofstr*. *Pump* may be used interactively and in pipelines. A common use is to get variable values into editor scripts. If \$a, \$b, and \$c have the values A, B, and C respectively, the two sequences below are equivalent:

<pre>pump -~   ed file 1,\$s/~a\$/~b/ ?~c? !</pre>	<pre>ed file 1,\$s/A\$/B/ ?C? q</pre>
--	---------------------------------------

The sequence above will work at the terminal as well as in Shell procedures. *Pump* is an efficient and convenient replacement for multiple uses of *echo(1)*; e.g., the following are equivalent:

<pre>pump &gt;file \$1 \$2 !</pre>	<pre>echo "\$1" &gt;file echo "\$2" &gt;&gt;file</pre>
------------------------------------	--

*Pump* is actually implemented inside the Shell, although it executes as a separate process.

## SEE ALSO

*echo(1)*, *sh(1)*

## BUGS

The size of *eofstr* is limited to 95 bytes, and it may not begin with '+'. .

**NAME**

`pwd` – working directory name

**SYNOPSIS**

`pwd`

**DESCRIPTION**

*Pwd* prints the pathname of the working (current) directory.

**SEE ALSO**

`chdir(1)`



**NAME**

quiz – test your knowledge

**SYNOPSIS**

**quiz** [ **-i** file ] [ **-t** ] [ category1 category2 ]

**DESCRIPTION**

*Quiz* gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, *quiz* gives instructions and lists the available categories.

*Quiz* tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **-t** flag specifies ‘tutorial’ mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **-i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```

line      = category newline | category ':' line
category  = alternate | category '[' alternate
alternate = empty | alternate primary
primary   = character | '[' category ']' | option
option    = '{' category '}'

```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash ‘\’ is used as with *sh*(I) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

**FILES**

/usr/lib/quiz/index  
 /usr/lib/quiz/\*

**NAME**

`rc` — Ratfor compiler

**SYNOPSIS**

`rc [ -c ] [ -r ] [ -f ] [ -v ] file ...`

**DESCRIPTION**

`Rc` invokes the Ratfor preprocessor on a set of Ratfor source files. It accepts three types of arguments:

Arguments whose names end with `‘.r’` are taken to be Ratfor source programs; they are preprocessed into Fortran and compiled. Each subroutine or function `‘name’` is placed on a separate file `name.f`, and its object code is left on `name.o`. The main routine is on `MAIN.f` and `MAIN.o`; block data subprograms go on `blockdata?.f` and `blockdata?.o`. The files resulting from a `‘.r’` file are loaded into a single object file `file.o`, and the intermediate object and Fortran files are removed.

The following flags are interpreted by `rc`. See `ld(I)` for load-time flags.

- `-c` Suppresses the loading phase of the compilation, as does any error in anything.
- `-f` Save Fortran intermediate files. This is primarily for debugging.
- `-r` Ratfor only; don't try to compile the Fortran. This implies `-f`.
- `-v` Don't list intermediate file names while compiling.

Arguments whose names end with `‘.f’` are taken to be Fortran source programs; they are compiled in the normal manner. (Only one Fortran routine is allowed in a `‘.f’` file.)

Other arguments are taken to be either loader flag arguments, or Fortran-compatible object programs, typically produced by an earlier `rc` run, or perhaps libraries of Fortran-compatible routines. These programs, together with the results of any compilations specified, are loaded to produce an executable program with name `a.out`.

**FILES**

<code>ratjunk</code>	temporary
<code>/usr/bin/ratfor</code>	preprocessor
<code>/usr/fort/fc1</code>	Fortran compiler

**SEE ALSO**

*RATFOR — A Preprocessor for a Rational Fortran* by B. W. Kernighan.  
`fc(I)` for Fortran error messages.

**DIAGNOSTICS**

Yes, both from `rc` itself and from Fortran.

**BUGS**

Limit of about 50 arguments, 10 block data files.

`#define` and `#include` lines in `‘.f’` files are not processed.

## NAME

reform – reformat text file

## SYNOPSIS

**reform** [*tabspec1* [*tabspec2*]] [+*bn*] [+*en*] [+*f*] [+*in*] [+*mn*] [+*pn*] [+*s*] [+*tn*]

## DESCRIPTION

*Reform* reads each line of the standard input file, reformats it, and then writes it to the standard output. Various combinations of reformatting operations can be selected, of which the most common involve rearrangement of tab characters. It is often used to trim trailing blanks, truncate lines to a specified length, or prepend blanks to lines.

*Reform* first scans its arguments, which may be given in any order. It then processes its input file, performing the following actions upon each line, in the order given:

- A line is read from the standard input.
- If +*s* is given, all characters up to the first tab are stripped off and saved for later addition to the end of the line. Presumably, these characters comprise an SCCS SID produced by *get(1)*.
- The line is expanded into a tabless form, by replacing tabs with blanks according to the *input* tab specification *tabspec1*.
- If +*pn* is given, *n* blanks are prepended to the line.
- If +*tn* is given, the line is truncated to a length of *n* characters.
- All trailing blanks are now removed.
- If +*en* is included, the line is extended out with blanks to the length of *n* characters.
- If +*s* is given, the previously-saved SCCS SID is added to the end of the line.
- If +*bn* is given, the *n* characters at the beginning of the line are converted to blanks, if and only if all of them are either digits or blanks.
- If +*mn* is included, the line is moved left, i.e., *n* characters are removed from the beginning of the line.
- The line is now contracted by replacing some blanks with tab characters according to the list of tabs indicated by the *output* tab specification *tabspec2*, and is written to the standard output file. Option +*i* controls the method of contraction (see below).

The various arguments accepted by *reform* are as follows:

*tabspec1* describes the tab stops assumed for the input file. This tab specification may take on any of the forms described in *tabs(1)*. In addition, the operand “--” indicates that the tab specification is to be found in the first line read from the standard input. If no legal tab specification is found there, -8 is assumed. If *tabspec1* is omitted entirely, “--” is assumed.

*tabspec2* describes the tabs assumed for the output file. It is interpreted in the same way as *tabspec1*, except that omission of *tabspec2* causes the value of *tabspec1* to be used for *tabspec2*.

The remaining arguments are all optional and may be used in any combination, although only a few combinations make much sense. Specifying an argument causes an action to be performed, as opposed to the usual default of not performing the action. Some options include numeric values, which also have default values. Option actions are applied to each line in the order described

above. Any line length mentioned applies to the length of a line just before the execution of the option described, and the terminating newline is never counted in the line length.

- +bn** causes the first *n* characters of a line to be converted to blanks, if and only if those characters include only blanks and digits. If *n* is omitted, the default value is 6, which is useful in deleting sequence numbers from COBOL programs.
- +en** causes each line shorter than *n* characters to be extended out with blanks to that length. Omitting *n* implies a default value of 72. This option is useful for those rare cases in which sequence numbers need to be added to an existing unnumbered file. The use of \$ in editor regular expressions is more convenient if all lines have equal length, so that the user can issue editor commands such as:

```
s/$/00001000/
```

- +f** causes a format line to be written to the standard output, preceding any other lines written. See *fspec(V)* for details regarding format specifications. The format line is taken from *tabspec2*, i.e., the line normally appears as follows:

```
<:t - tabspec2 d:>
```

If *tabspec2* is of the form `--filename` (i.e., an indirect reference to a tab specification in the first line of the named file), then that tab specification line is written to the standard output.

- +in** controls the technique used to compress interior blanks into tabs. Unless this option is specified, any sequence of 1 or more blanks may be converted to a single tab character if that sequence occurs just before a tab stop. This causes no problems for blanks that occur before the first nonblank character in a line, and it is always possible to convert the result back to an equivalent tabless form. However, occasionally an interior blank (one occurring after the first nonblank) is converted to a tab when this is not intended. For instance, this might occur in any program written in a language utilizing blanks as delimiters. Any single blank might be converted to a tab if it occurred just before a tab stop. Insertion or deletion of characters preceding such a tab may cause it to be interpreted in an unexpected way at a later time. If the **+i** option is used, no string of blanks may be converted to a tab unless there are *n* or more contiguous blanks. The default value is 2. Note that leading blanks are always converted to tabs when possible. It is recommended that conversion of programs from non-PWB to PWB systems use this option.
- +mn** causes each line to be moved left *n* characters, with a default value of 6. This can be useful for crunching COBOL programs.
- +pn** causes *n* blanks to be prepended (default of 6 if *n* is omitted). This option is effectively the inverse of **+mn**, and is often useful for adjusting the position of *nroff(I)* output for terminals lacking both forms tractor positioning and a settable left margin.
- +s** is used with the **-m** option of *ger(I)*. The **-m** option causes *ger* to prepend to each generated line the appropriate SCCS SID, followed by a tab. The **+s** option causes *reform* to remove the SID from the front of the line, save it, then add it later to the end of the line. Because **+e72** is implied by this option, the effect is to produce 80-character card images with SCCS SID in columns 73-80. Up to 8 characters of the SID are shown; if it is longer, the eighth character is replaced by '\*' and any characters to the right of it are discarded.
- +tn** causes any line longer than *n* characters to be truncated to that length. If *n* is omitted, the length defaults to 72. Sequence numbers can thus be removed and any blanks immediately preceding them deleted.

The following illustrate typical uses of *reform*. The terms "PWB" and "OBJECT" below refer to UNIX and non-UNIX computer systems, respectively. Each arrow indicates the direction of conversion. The character '?' indicates an arbitrary tab specification; see *tabs(1)* for descriptions of legal specifications.

OBJECT ----> PWB (i.e., manipulation of RJE output):

Note that files transferred by RJE from OBJECT to PWB materialize with format **-8**.

`reform -8 ? +t +f <oldfile >newfile` (into arbitrary format)

`reform -8 -c +t +b +i <oldfile >newfile` (into COBOL)

`reform -8 -c3 +t +m +i <oldfile >newfile` (into COBOL, crunched)

NOTE: `-c3` is the preferred format for COBOL; it uses the least disk space of the COBOL formats.

PWB ----> OBJECT (i.e., preparation of files for RJE submission):

`reform ? -8 <oldfile >newfile` (from arbitrary format into **-8**)

`get -p -m sccsfile | reform +s | send ...`

PWB ONLY (i.e., no involvement with other systems):

`pr file | reform ? -0 <oldfile` (print on terminal without hardware tabs)

`reform ? -0 <oldfile >newfile` (convert file to tabless format)

#### DIAGNOSTICS

All diagnostics are fatal, and the offending line is displayed following the message.

"line too long" a line exceeds 512 characters (in tabless form).

"not SCCS -m" a line does not have at least one tab when `+s` flag is used.

Any of the diagnostics of *tabs(1)* can also appear.

#### EXIT CODES

0 - normal

1 - any error

#### SEE ALSO

`fspec(V)`, `get(1)`, `nroff(1)`, `send(1)`, `tabs(1)`

#### BUGS

*Reform* is aware of the meanings of backspaces and escape sequences, so that it can be used as a postprocessor for *nroff*. However, be warned that the `+e`, `+m`, `+t` options only count characters, not positions. Anyone using these options on output containing backspaces or halfline motions will probably obtain unexpected results.

**NAME**

`regcmp` – regular expression compile

**SYNOPSIS**

`regcmp [-] file ...`

**DESCRIPTION**

*Regcmp*, in most cases, precludes the need for calling *regcmp* (see *regex(III)*) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file.i*. If the “-” option is used, the output will be placed in *file.c*.

The format of entries in *file* is a name (C variable), followed by one or more blanks, followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source, which declares each variable name as an *extern char* array, and initializes that array with the compiled form of the corresponding regular expression. *File.i* files may thus be *included* into C programs, or *file.c* files may be compiled and later loaded. Diagnostics are self-explanatory.

**Example:**

```

name      "([A-Za-z][A-Za-z0-9_]*)$0"
telno     "\\({0,1}[2-9][01][1-9])$0\\){0,1}*"
          "([2-9][0-9]{2})$1[-]{0,1}"
          "([0-9]{4})$2"

```

In the C program which uses the *regcmp* output,

```
    regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

**SEE ALSO**

*regex(III)*

**NAME**

**rgrep** - search a file for a pattern

**SYNOPSIS**

**rgrep** [ **-v** ] [ **-b** ] [ **-c** ] [ **-n** ] *expression* [ *file* ] ...

**DESCRIPTION**

*Rgrep* is an extended form of *grep* which uses the facilities of the *regex(III)* routine. *Rgrep* searches the input files (standard input default) for lines matching the regular expression. Normally, each line found is copied to the standard output. If the **-v** flag is used, all lines but those matching are printed. If the **-c** flag is used, only a count of matching lines is printed. If the **-n** flag is used, each line is preceded its relative line number in the file. If the **-b** flag is used, each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

In all cases the file name is shown if there is more than one input file.

For a complete description of the regular expression, see *ed(I)* and *regex(III)*. Care should be taken when using the characters \$ \* [ ^ | ( ) and \ in the regular expression as they are also meaningful to the Shell. It is generally necessary to enclose the entire *expression* argument in quotes.

**SEE ALSO**

*ed(I)*, *sh(I)*, *regex(III)*

## NAME

rjestat – RJE status and enquiries

## SYNOPSIS

rjestat [ - ] [ A ] [ B ] [ 1110 ]

## DESCRIPTION

When invoked without the ‘-’ argument, *rjestat* reports the current status of RJE links to the specified host computers. When invoked with the ‘-’ argument, *rjestat* sets up an interactive status terminal. If no hosts are cited explicitly, the specification defaults to all those for which a given PWB/UNIX is configured. The “host” pseudonyms **A**, **B**, and **1110** are built into the RJE software. **A** and **B** may be used to represent any IBM host machine. Their actual destinations are immaterial to RJE. The pseudonym **1110** is built into RJE to represent any UNIVAC host.

To enter an enquiry via such a status terminal, you must first generate an interrupt. This can be done by hitting the DEL key or the BREAK/INTERRUPT key. *Rjestat* will respond by prompting for enquiries directed to each host in turn. The line on which a prompt appears may be completed to form a legitimate display command for that particular host. If the line is terminated with a ‘\’, the prompt will be repeated, otherwise it will advance to the next host. A carriage return alone indicates that no enquiry is to be directed to a particular host. You should expect to wait at least 30 seconds for a response.

An interrupt will temporarily halt the display of responses. It can therefore be used to inhibit roll-up on a CRT terminal. The display of responses will resume after all prompts have been satisfied (perhaps by null completions).

To exit from the status terminal, generate a quit signal or type DEL followed by EOT.

The UNIVAC 1110 capability is only supported at the BTL Piscataway location.

## FILES

/dev/rje*	DQS-11's used by RJE
/usr/rje/sys	PWB/UNIX system name
/usr/rje/lines	configuration table

And, in the directory for each RJE subsystem:

log	activity log
resp	concatenated responses
status	message of the day
xmit*	files queued
*mesg	enquiry slot
*init	boot program

## SEE ALSO

*Guide to IBM Remote Job Entry for PWB/UNIX Users* by A. L. Sabsevit.   
*OS/VS2 HASP II Version 4 Operator's Guide*, IBM SRL #GC27-6993.   
*Operator's Library: OS/VS2 Reference (JES2)*, IBM SRL #GC38-0210.



**NAME**

**rm** - remove (unlink) files

**SYNOPSIS**

**rm** [ **-f** ] [ **-r** ] name ...

**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 the user does not have write permission on a file, *rm* prints the file name and its mode, then reads a line from the standard input. If the line begins with **y**, the file is removed, otherwise it is not. The question is not asked if option **-f** was given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory. To remove directories *per se* see *rmdir*(1).

**FILES**

/etc/glob to implement the **-r** flag

**SEE ALSO**

*rmdir*(1)

**BUGS**

When *rm* removes the contents of a directory under the **-r** flag, full pathnames are not printed in diagnostics.

**NAME**

`rmdel` – remove a delta from an SCCS file

**SYNOPSIS**

`rmdel` **--rSID** name ...

**DESCRIPTION**

*Rmdel* removes the delta specified by the SID from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file.

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with "s."), and unreadable files, are silently ignored. If a name of "--" is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files, and unreadable files, are silently ignored.

The exact permissions necessary to remove a delta are documented in the *SCCS/PWB User's Manual*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

**FILES**

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

**SEE ALSO**

*get*(1), *delta*(1), *prt*(1), *help*(1), *scsfile*(V)  
*SCCS/PWB User's Manual* by L. E. Bonanni and A. L. Glasser

**DIAGNOSTICS**

Use *help*(1) for explanations.

**NAME**

`rmdir` - remove directory

**SYNOPSIS**

`rmdir` dir ...

**DESCRIPTION**

*Rmdir* removes (deletes) directories. The directory must be empty (except for the standard entries `.` and `..`, which *rmdir* itself removes). Write permission is required in the directory in which the directory to be removed appears.

**BUGS**

Needs a `-r` flag.

Actually, write permission in the directory's parent is *not* required.

Mildly unpleasant consequences can follow removal of your own or someone else's current directory.

## NAME

roff — format text

## SYNOPSIS

roff [ +n ] [ -n ] [ -s ] [ -h ] file ...

## DESCRIPTION

*Roff* formats text according to control lines embedded in the text in the given files. Encountering a nonexistent file terminates printing. Incoming interconsole messages are turned off during printing. The optional flag arguments mean:

- +n Start printing at the first page with number *n*.
- n Stop printing at the first page numbered higher than *n*.
- s Stop before each page (including the first) to allow paper manipulation; resume on receipt of an interrupt signal.
- h Insert tabs in the output stream to replace spaces whenever appropriate.

Input consists of intermixed *text lines*, which contain information to be formatted, and *request lines*, which contain instructions about how to format it. Request lines begin with a distinguished *control character*, normally a period.

Output lines may be *filled* as nearly as possible with words without regard to input lineation. Line *breaks* may be caused at specified places by certain commands, or by the appearance of an empty input line or an input line beginning with a space.

The capabilities of *roff* are specified in the attached Request Summary. Numerical values are denoted there by *n* or +*n*, titles by *t*, and single characters by *c*. Numbers denoted +*n* may be signed + or -, in which case they signify relative changes to a quantity, otherwise they signify an absolute resetting. Missing *n* fields are ordinarily taken to be 1, missing *t* fields to be empty, and *c* fields to shut off the appropriate special interpretation.

Running titles usually appear at top and bottom of every page. They are set by requests like

```
.he 'part1'part2'part3'
```

Part1 is left justified, part2 is centered, and part3 is right justified on the page. Any % sign in a title is replaced by the current page number. Any nonblank may serve as a quote.

ASCII tab characters are replaced in the input by a *replacement character*, normally a space, according to the column settings given by a .ta command. (See .tr for how to convert this character on output.)

Automatic hyphenation of filled output is done under control of .hy. When a word contains a designated *hyphenation character*, that character disappears from the output and hyphens can be introduced into the word at the marked places only.

## FILES

```
/usr/lib/suftab  suffix hyphenation tables
/tmp/rtn?      temporary
```

## SEE ALSO

nroff(I), troff(I)

## BUGS

*Roff* is the simplest of the run-off programs, but is utterly frozen and quite obsolescent.

## REQUEST SUMMARY

<i>Request</i>	<i>Break</i>	<i>Initial</i>	<i>Meaning</i>
.ad	yes	yes	Begin adjusting right margins.
.ar	no	arabic	Arabic page numbers.
.br	yes	-	Causes a line break — the filling of the current line is stopped.
.bl n	yes	-	Insert of n blank lines, on new page if necessary.
.bp +n	yes	n=1	Begin new page and number it n; no n means '+1'.
.cc c	no	c=.	Control character becomes 'c'.
.ce n	yes	-	Center the next n input lines, without filling.
.de xx	no	-	Define parameterless macro to be invoked by request '.xx' (definition ends on line beginning '..').
.ds	yes	no	Double space; same as '.ls 2'.
.ef t	no	t=****	Even foot title becomes t.
.eh t	no	t=****	Even head title becomes t.
.fi	yes	yes	Begin filling output lines.
.fo	no	t=****	All foot titles are t.
.hc c	no	none	Hyphenation character becomes 'c'.
.he t	no	t=****	All head titles are t.
.hx	no	-	Title lines are suppressed.
.hy n	no	n=1	Hyphenation is done, if n=1; and is not done, if n=0.
.ig	no	-	Ignore input lines through a line beginning with '..'.
.in +n	yes	-	Indent n spaces from left margin.
.ix +n	no	-	Same as '.in' but without break.
.li n	no	-	Literal, treat next n lines as text.
.ll +n	no	n=65	Line length including indent is n characters.
.ls +n	yes	n=1	Line spacing set to n lines per output line.
.m1 n	no	n=2	Put n blank lines between the top of page and head title.
.m2 n	no	n=2	n blank lines put between head title and beginning of text on page.
.m3 n	no	n=1	n blank lines put between end of text and foot title.
.m4 n	no	n=3	n blank lines put between the foot title and the bottom of page.
.na	yes	no	Stop adjusting the right margin.
.ne n	no	-	Begin new page, if n output lines cannot fit on present page.
.nn +n	no	-	The next n output lines are not numbered.
.n1	no	no	Add 5 to page offset; number lines in margin from 1 on each page.
.n2 n	no	no	Add 5 to page offset; number lines from n; stop if n=0.
.ni +n	no	n=0	Line numbers are indented n.
.nf	yes	no	Stop filling output lines.
.nx filename		-	Change to input file 'filename'.
.of t	no	t=****	Odd foot title becomes t.
.oh t	no	t=****	Odd head title becomes t.
.pa +n	yes	n=1	Same as '.bp'.
.pl +n	no	n=66	Total paper length taken to be n lines.
.po +n	no	n=0	Page offset. All lines are preceded by n spaces.
.ro	no	arabic	Roman page numbers.
.sk n	no	-	Produce n blank pages starting next page.
.sp n	yes	-	Insert block of n blank lines, except at top of page.
.ss	yes	yes	Single space output lines, equivalent to '.ls 1'.
.ta n n..		-	Pseudotab settings. Initial tab settings are columns 9 17 25 ...
.tc c	no	space	Tab replacement character becomes 'c'.
.ti +n	yes	-	Temporarily indent next output line n spaces.
.tr cdef..	no	-	Translate c into d, e into f, etc.
.ul n	no	-	Underline the letters and numbers in the next n input lines.

**NAME**

`rsh` — restricted shell (command interpreter)

**SYNOPSIS**

`rsh [ -x ] [ - ] [ -ct ] [ name [ arg1 ... ] ]`

**DESCRIPTION**

*Rsh* is a restricted version of the standard command interpreter *sh(1)*. It is used to set up login names or execution environments whose capabilities are more controlled than that of the standard shell. The actions of *rsh* are identical to those of *sh*, except for the following restrictions:

- 1) *chdir* is not allowed.
- 2) changes to the shell variable '\$p' are not permitted.
- 3) it is illegal to use '/' in the name of a command.
- 4) *next* is not permitted.
- 5) '>' and '>>' are disallowed.

These restrictions combine to lock a user into the login directory, limit the set of invocable commands to those found in directories included in the '.path' file, and eliminate the direct creation or modification of files. When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to write shell procedures using the full power of the standard shell, while the end user is restricted to a limited menu of commands.

*Rsh* is actually just a link to *sh*.

**FILES**

/etc/glob, which interprets '\*', '?', and '['.  
/dev/null as a source of end-of-file.  
.path in login directory to initialize \$p.  
.profile in login directory for general initialization.  
/etc/sha for accounting information.

**SEE ALSO**

*sh(1)*

**BUGS**

It would be better to have a flag for *opt* which changed *sh* into *rsh* dynamically. With a non-interruptable '.profile', it would be possible to act as *sh*, use *chdir* (for example), and then change into *rsh* at the end of initialization.

**NAME**

`sccsdiff` – compare two versions of an SCCS file

**SYNOPSIS**

`sccsdiff` *old-spec* *new-spec* [ *pr-args* ] *sccsfile* ...

**DESCRIPTION**

*Sccsdiff* compares two versions of an SCCS file and generates the differences between the two versions. The *old-spec* is any valid *get(I)* specifier (e.g., `-r1.1`) for the old version to be gotten. Similarly, *new-spec* is any valid *get(I)* specifier (e.g., `-r1.4`) for the new version to be gotten. The *pr-args* are any valid *pr(I)* arguments which begin with a “-”, except for “-h” (the output of *sccsdiff* is piped through *pr(I)*). Any number of SCCS files may be specified, but the *old-spec* and *new-spec* apply to all files.

*Sccsdiff* is a simple shell procedure; interested persons should “`cat /usr/bin/sccsdiff`” to discover how it works.

**FILES**

`/tmp/get?????` temporary for old gotten version  
`/usr/bin/bdiff` program that generates differences

**SEE ALSO**

*get(I)*, *help(I)*, *pr(I)*, *bdiff(I)*  
*SCCS/PWB User's Manual* by L. E. Bonanni and A. L. Glasser.

**DIAGNOSTICS**

Use *help(I)* for explanations.

## NAME

sed – stream editor

## SYNOPSIS

sed [ **-g** ] [ **-n** ] [ **-f** commandfile ] ... [ [ **-e** ] command ] ... [ file ] ...

## DESCRIPTION

*Sed* copies the input *files* (default is standard input) to the standard output, perhaps performing one or more editor commands (see *ed(1)*) on each line.

The **-g** flag indicates that all *s* commands should be executed as though followed by a *g*. If only some substitutions are to be done globally, leave out the **-g** flag and put the *g*'s at the end of the appropriate command lines.

The **-n** flag indicates that only lines that are explicitly printed by *p* commands are to be copied to the standard output. In order to avoid getting double copies of some lines in the standard output, the *p* command is ignored unless the **-n** flag is set.

The **-e** flag indicates that the next argument is an editor command.

The **-f** flag indicates that the next argument is a file name; the file contains editor commands, one to a line. Commands that are inherently multi-line, like *a* or *c*, should have the interior new-lines escaped by '\'. Append, insert, and change modes are terminated by a non-escaped new-line.

The **-e** and **-f** flags may be intermixed in any order.

If no **-e** or **-f** flags are given, the first argument is taken by default to be an editor command.

Addresses are allowed. The meaning of *two* addresses is: "Attempt this command on the first line that matches the first address, and on all subsequent lines up to and including the first subsequent line that matches the second address; then search for a match of the first address and iterate." *One* address means: "Attempt this command on all lines that match the address." Either line-numbers or regular expressions are allowed as addresses. Line numbers increase monotonically throughout *all* the input files, so that, if *n* is the number of the last line of the first input file, then *n+1* is the number of the first line of the second file, etc. A '\$' as an address matches the *last* line of the *last* input file.

The intention is to simulate the editor as exactly as possible, but the line-at-a-time operation makes certain differences unavoidable or desirable:

1. There is no notion of '.' and no relative addressing.
2. Commands with no addresses are defaulted to *1,\$* rather than to dot.
3. Addresses specified as regular expressions must be delimited by '/'; '?' is an error.
4. Expressions in addresses are not allowed (i.e., '+', '-').
5. Commands may have only as many addresses as they can use. That is, no command may have more than two addresses; the *a*, *i*, and *r* commands may have only one address.
6. A *p* at the end of a command only works with the *s* command. For other commands, or if the **-n** flag is not in effect, a *p* at the end of a command line is ignored.
7. A *w* may appear at the end of a *s* command. It should be followed by a single space and a file name. If the *s* command succeeds, the modified line is appended to the file. All files are opened when the commands are being compiled, and closed when the program terminates. Only ten distinct file names may appear in *w* commands in a single execution of



*sed*. Unlike *p*, *w* takes effect regardless of the *-n* flag. If both *p* and *w* are appended to the same substitute command, they must be in the order *pw*.

8. The only editor commands available are *a*, *c*, *d*, *i*, *s*, *p*, *q*, *r*, *w*, *g*, *v*, and *=*. A successful execution of a *q* command causes the current line to be written out if it should be, and execution terminated. When a line is deleted by a *d* or *c* command, no further commands are attempted on its corpse, but another line is immediately read from the input (but see item 10. below).
9. The *next* line command, *n*, replaces the current line by the next line from the input file. The list of editing commands is continued after the *n* command is executed.
10. If an *a*, *i*, or *r* command is successfully executed, the text is inserted into the standard output whether or not the line on which the match was made is later deleted or not. Thus the commands:

```

        /b/a\
        XXX
        /b/,/c/d
applied to the file
        a
        b
        c
        d
will produce
        a
        XXX
        d

```

on the output.

11. Text inserted in the output stream by the *a*, *i*, *c*, or *r* commands is not scanned for any pattern matches, nor are any editor commands applied to it.

*Sed* supports three commands to control the flow of processing. These commands do no editing on the input line, but serve to control the order in which multiple editing commands are applied to an input line.

12. The label command, *: label*, marks a place in the list of editing commands which may be referred to by *j* and *t* commands (see 13. and 14. below); the *label* may be any sequence of eight or fewer characters; if two different colon commands have identical labels, a compile-time diagnostic will be generated and no execution attempted.
13. The jump command, *j label*, causes the sequence of editing commands being applied to the current input line to be restarted immediately after the place where a colon command with the same *label* was encountered. If no colon command with the same label can be found after all editing commands have been compiled, a compile-time diagnostic is produced and no execution is attempted. A *j* command with no *label* is taken to be a jump to the end of the list of editing commands; whatever should be done with the current input line is done, and another input line is read; the list of editing commands is restarted from the beginning of that line.
14. The test command, *t label*, tests whether *any* successful substitutions have been made on the current input line; if so, it jumps to *label*; if not, it does nothing. The flag that indicates that a successful substitution has occurred on the current input line is reset by either reading a new line or by executing the *t* command.

*Sed* also supports command grouping and several operations that can build lines into a pattern space to be operated upon by other commands.

- Commands may be grouped by curly braces. The opening brace must appear in the place where a command would ordinarily appear; the closing brace must appear on a line by itself (except for leading blanks or tabs). If the first line of a command file has *#n* as its first two characters, the no-copy flag is set, as though the *-n* option had been given on the command line. The remainder of this first line is ignored and may be used for a title or a comment. As an example:

```
#n Print first non-blank line after a blank line, and first line, if non-blank.
1{
    ./p
}
/^$/{
: loop
    n
    ./{
        p
    }
    j loop
}
```

- The *Next* command, *N*, appends the next input line to the current line; the two lines are separated by a new-line character, that may be matched by *'\n'*.
- The *Delete* command, *D*, deletes up to and including the first (leftmost) new-line in the current pattern space. If the pattern space becomes empty (the only new-line is at the end of the space), *Delete* reads another line from the input. The list of editing commands is restarted from the beginning.
- The *Print* command, *P*, prints on standard output up to and including the first new-line in the pattern space.

SEE ALSO  
ed(I)

#### BUGS

Lines are silently truncated to a maximum length of 512 characters. The “plus”, “range”, and “through” regular expression operators (“+”, “\{ }”, “[ - ]”) of *ed(I)* are not implemented in *sed*.

## NAME

send — submit RJE job

## SYNOPSIS

send argument ...

## DESCRIPTION

*Send* is a command-level interface to the RJE subsystems *hasp*(VIII) and *uvac*(VIII). It allows the user to collect input from various sources in order to create a run stream consisting of card images. *Send* creates a temporary file, with a special format, to contain the collected run stream, and then queues the file for transmission by invoking *haspqr* or *uvacqr*, as appropriate. Further processing of the job is controlled by the appropriate PWB/UNIX RJE subsystem and the host computer to which the job is submitted.

Possible sources of input to *send* are: ordinary files, standard input, the terminal, and the output of a command or shell file. Each source of input is treated as a virtual file, and no distinction is made based upon its origin. Typical input is an ASCII text file of the sort that is created by the editor *ed*(1). An optional format specification appearing in the first line of a file (see *fspec*(V)) determines the settings according to which tabs are expanded into spaces. In addition, lines that begin with “~” are normally interpreted as commands controlling the execution of *send*. They may be used to set or reset flags, to define keyword substitutions, and to open new sources of input in the midst of the current source. Other text lines are translated one-for-one into card images of the run stream.

The run stream that results from this collection is treated as one job by the RJE subsystems. *Send* provides a card count for the run stream, and the queuer that is invoked announces the position that the job has been assigned in the queue of jobs waiting to be transmitted. The initial card of a job submitted to an IBM system must have a “/” in the first column. The initial card of a job submitted to a UNIVAC system must begin with a “@RUN” or “run”, etc. Any cards preceding these will be excised. If a host computer is not specified before the first card of the runstream is ready to be sent, *send* will select a reasonable default. In the case of an IBM job, all cards beginning “/\*\$” will be excised from the runstream, because they are HASP command cards.

The arguments that *send* accepts are described below. An argument is interpreted according to the first pattern that it matches. Preceding a character with “\” causes it to lose any special meaning it might otherwise have when matching against an argument pattern.

.	Close the current source.
-	Open standard input as a new source.
+	Open the terminal as a new source.
:spec:	Establish a default format specification for included sources, e.g., :m6t-12:.
:message	Print message on the terminal.
-:prompt	Open standard input and, if it is a terminal, print <i>prompt</i> .
+:prompt	Open the terminal and print <i>prompt</i> .
-flags	Set the specified flags, which are described below.
+flags	Reset the specified flags.

<i>=flags</i>	Restore the specified flags to their state at the previous level.
<i>!command</i>	Execute the specified PWB/UNIX <i>command</i> via the one-line Shell, with input redirected to <i>/dev/null</i> as a default. Open the standard output of the command as a new source.
<i>\$line</i>	Collect contiguous arguments of this form and write them as consecutive lines to a temporary file; then have the file executed by the Shell. Open the standard output of the Shell as a new source.
<i>~comment</i>	Ignore this argument.
<i>=:keyword</i>	Prompt for a definition of <i>keyword</i> from the terminal.
<i>keyword=^xx</i>	Define <i>keyword</i> as a two-digit hexadecimal character code.
<i>keyword=string</i>	Define <i>keyword</i> in terms of a replacement string.
<i>host</i>	Job is to be submitted to: <b>A</b> , <b>B</b> , <b>1110</b> . The pseudonyms <b>A</b> and <b>B</b> are built into RJE to represent any IBM host connection. Their actual destinations are immaterial to RJE. The pseudonym <b>1110</b> is built into RJE to represent any UNIVAC host.
<i>filename</i>	Open the specified file as a new source of input.

Arguments of the form “!*chdir directory*” will be trapped so that the *send* process can execute the specified *chdir* itself. The original directory will be restored at the end of any source that contains a *chdir*.

The flags recognized by *send* are described in terms of the special processing that occurs when they are set:

- l List card images on standard output. EBCDIC characters are translated back to ASCII.
- q Do not output card images.
- f Do not fold lower case to upper.
- t Trace progress on diagnostic output, by announcing the opening of input sources.
- k Ignore the keywords that are active at the previous level and erase any keyword definitions that have been made at the current level.
- r Process included sources in raw mode; pack arbitrary 8-bit bytes one per column (80 columns per card) until an end-of-file.
- i Do not interpret control lines in included sources; treat them as text.
- s Make keyword substitutions before detecting and interpreting control lines.
- y Suppress error diagnostics and submit job anyway.
- g Gather mode, qualifying -l flag; list text lines before converting them to card images.
- h Write listing with standard tabs.
- p Prompt with “\*” when taking input from the terminal.
- m When input returns to the terminal from a lower level, repeat the prompt, if any.
- a Make -k flag propagate to included sources, thereby protecting them from keyword substitutions.

- c List control lines on diagnostic output.
- d Extend the current set of keyword definitions by adding those active at the end of included sources.

Control lines are input lines that begin with “~”. In the default mode **+ir**, they are interpreted as commands to *send*. Normally they are detected immediately and read literally. The **-s** flag forces keyword substitutions to be made before control lines are intercepted and interpreted. Arguments appearing in control lines are handled exactly like the command arguments to *send*, except that they are processed at a nested level of input.

The two possible formats for a control line are: “~argument” and “~ argument ...”. In the first case, where the “~” is not followed by a space, the remainder of the line is taken as a single argument to *send*. In the second case, the line is parsed to obtain a sequence of arguments delimited by spaces. In this case the quotes “” and “” may be employed to pass embedded spaces.

The interpretation of the argument “.” is chosen so that an input line consisting of “.” is treated as a logical end-of-file. The following example illustrates some of the above conventions:

```
send -
~ argument ...
~
```

This sequence of three lines is equivalent to the command synopsis at the beginning of this description. In fact, the “-” is not even required. By convention, the *send* command reads standard input if no other input source is specified. *Send* may therefore be employed as a filter with side-effects.

The execution of the *send* command is controlled at each instant by a current environment, which includes the format specification for the input source, a default format specification for included sources, the settings of the mode flags, and the active set of keyword definitions. This environment can be altered dynamically. When a control line opens a new source of input, the current environment is pushed onto a stack, to be restored when input resumes from the old source. The initial format specification for the new source is taken from the first line of the file. If none is provided, the established default is used or, in its absence, standard tabs. The initial mode settings and active keywords are copied from the old environment. Changes made while processing the new source will not affect the environment of the old source, with one exception: if **-d** mode is set in the old environment, the old keyword context will be augmented by those definitions that are active at the end of the new source. When *send* first begins execution, all mode flags are reset, and no keywords are defined.

The initial, reset state for all mode flags is the “+” state. In general, special processing associated with a mode *x* is invoked by flag **-x** and is revoked by flag **+x**. Most mode settings have an immediate effect on the processing of the current source. Exceptions to this are the **-r** and **-i** flags, which apply only to included source, causing it to be processed in an uninterpreted manner.

A keyword is an arbitrary ASCII string for which a replacement has been defined. The replacement may be another string, or (for IBM RJE only) the hexadecimal code for a single 8-bit byte. At any instant, a given set of keyword definitions is active. Input text lines are scanned, in one pass from left to right, and longest matches are attempted between substrings of the line and the active set of keywords. Characters that do not match are output, subject to folding and the standard translation. Keywords are replaced by the specified hexadecimal code or replacement string, which is then output character by character. The expansion of tabs and length checking, according to the format specification of an input source, are delayed until substitutions have been made in a line.

All of the keywords definitions made in the current source may be deleted by setting the `-k` flag. It then becomes possible to reuse them, although this is not recommended. Setting the `-k` flag also causes keyword definitions active at the previous source level to be ignored. Setting the `+k` flag causes keywords at the previous level to be ignored but does not delete the definitions made at the current level. The `=k` argument reactivates the definitions of the previous level.

A keyword may not be redefined, except redundantly, if it is active at some level of source input and its replacement is not null. Prompts for keywords that have already been defined at some higher level will simply cause the definitions to be copied down to the current level; new definitions will not be solicited. Only in the case where a keyword is defined by a null replacement, `A=`, is a redefinition allowed, `A=a`. Prompts for the keyword, `=:A`, will be satisfied by either definition.

Keyword substitution is an elementary macro facility that is easily explained and that appears useful enough to warrant its inclusion in the `send` command. More complex replacements are the function of a general macro processor (`m4(I)`), perhaps. To reduce the overhead of string comparison, it is recommended that keywords be chosen so that their initial characters are unusual. For example, let them all be upper case.

`Send` performs two types of error checking on input text lines. Firstly, only ASCII graphics and tabs are permitted in input text. Secondly, the length of a text line, after substitutions have been made, may not exceed 80 bytes for IBM, or 132 bytes for UNIVAC. The length of each line may be additionally constrained by a size parameter in the format specification for an input source. Diagnostic output provides the location of each erroneous line, by line number and input source, a description of the error, and the card image that results. Other routine errors that are announced are the inability to open or write files, and abnormal exits from the Shell. Normally, the occurrence of any error causes `send`, before invoking the queuer, to prompt for positive affirmation that the suspect run stream should be submitted.

The `hasp` subsystem, which supports IBM RJE, operates in EBCDIC code. The `send` command is therefore required to translate ASCII characters into their EBCDIC equivalents. The standard conversion is based on the character set described in "Appendix H" of *IBM System/370 Principles of Operation* (IBM SRL GA22-7000). Each ASCII character in the octal range 040-176 possesses an EBCDIC graphic equivalent into which it is mapped, with four exceptions: broken vertical bar into "I", "^" into "~", "[" into "c", "]" into broken vertical bar. In listings requested from `send` and in printed output returned by `hasp`, the reverse translation is made from EBCDIC to ASCII, with the qualification that EBCDIC codes that do not have ASCII equivalents are translated into "". The `uvac` subsystem, on the other hand, operates in ASCII code, and any translations between ASCII and field-data are made, in accordance with the UNIVAC standard, by the host computer.

Additional control over the translation process is afforded by the `-f` flag and hexadecimal character codes. As a default, `send` folds lower-case letters into upper case. For UNIVAC RJE it does more: the entire ASCII range 140-176 is folded into 100-136, so that "", for example, becomes "@". In either case, setting the `-f` flag inhibits any folding. Non-standard character codes are obtained as a special case of keyword substitution.

When invoked under the name `gath`, the `send` command establishes initial flag settings `-lgq` and suppresses announcement of a zero card count. While in `-gq` mode, long lines that are detected elicit a diagnostic but are not truncated. Also, in this mode, it is potentially useful to convey non-graphics to standard output. To prevent `gath` from deleting non-printing characters, each may be declared as a single character keyword whose replacement is itself. To retain backspaces, for example, supply the argument "BS=BS", where BS denotes the ASCII character whose octal code is 010.

The UNIVAC 1110 capability is only supported at the BTL Piscataway location.

#### FILES

/bin/sh	Shell
/tmp/sh*	Shell temporary
/usr/rje/sys	PWB/UNIX system name, e.g., "A"
/usr/rje/lines	RJE configuration table

And, where *xxxx* is either *hasp* or *uvac*:

/usr/xxxx/pool/stm*	temporary
/usr/xxxx/xmit???	queued output
/usr/xxxx/xxxxqer	queueing program
/usr/xxxx/xxxxlock	null file for lockout
/usr/xxxx/xxxxstat	queue status record

#### SEE ALSO

help(I), m4(I), sh(I), ascii(V), ebcdic(V), fspec(V), hasp(VIII)  
*Guide to IBM Remote Job Entry for PWB/UNIX Users* by A. L. Sabsevitz.

#### DIAGNOSTICS

"non-graphic deleted", "undefined tab deleted", "long line detected", "long line truncated",  
 "illegal card excised" – followed by the resulting card image.  
 "Errors detected" – type "y" to submit anyway.

Use *help(I)* for explanations of error messages.

#### BUGS

Standard input is read in blocks, and unused bytes are returned via *seek(II)*. If standard input is a pipe, multiple arguments of the form "-" and "-:prompt" should not be used, nor should the logical end-of-file ".".

## NAME

sh — shell (command interpreter)

## SYNOPSIS

sh [ -v ] [ - ] [ -ct ] [ name [ arg1 ... ] ]

## DESCRIPTION

*Sh* is the standard command interpreter. It is the program which reads and arranges the execution of the command lines typed by most users. It may itself be called as a command to interpret files of commands. Before discussing the arguments to the Shell when it is used as a command, the structure of command lines themselves will be given.

**Commands.** Each command is a sequence of non-blank command arguments separated by blanks. The first argument specifies the name of a command to be executed. Except for certain types of special arguments discussed below, the arguments other than the command name are passed without interpretation to the invoked command.

By default, if the first argument is the name of an executable file, it is invoked; otherwise the string “/bin/” is prepended to the argument. (In this way most standard commands, which reside in “/bin”, are found.) If no such command is found, the string “/usr” is further prepended (to give “/usr/bin/command”) and another attempt is made to execute the resulting file. (Certain lesser-used commands live in “/usr/bin”.) If a command name contains a “/”, it is invoked as is, and no prepending ever occurs. This standard command search sequence may be changed by the user. See the description of the Shell variable “\$p” below.

If a non-directory file exists that matches the command name and has executable mode, but not the form of an executable program (does not begin with the proper magic number) then it is assumed to be an ASCII file of commands and a new Shell is created to execute it. See “Argument passing” below.

If the file cannot be found, a diagnostic is printed.

**Command lines.** One or more commands separated by “|” or “” constitute a chain of *filters*, or a *pipeline*. The standard output of each command but the last is taken as the standard input of the next command. Each command is run as a separate process, connected by pipes (see *pipe*(II)) to its neighbors. A command line contained in parentheses “( )” may appear in place of a simple command as a filter.

A *command line* consists of one or more pipelines separated, and perhaps terminated by “;” or “&”, or separated by “|” or “&&”. The semicolon designates sequential execution. The ampersand causes the following pipeline to be executed without waiting for the preceding pipeline to finish. The process id of the preceding pipeline is reported, so that it may be used if necessary for a subsequent *wait* or *kill*. A pipeline following “&&” is executed only if the preceding pipeline completed successfully (exit code zero), while that following “|” is executed only if the preceding one did *not* execute successfully (exit code non-zero). The exit code tested is that of the last command in the pipeline. The “&&” operator has higher precedence.

**Termination Reporting.** If a command (not followed by “&”) terminates abnormally, a message is printed. (All terminations other than exit and interrupt are considered abnormal). Termination reports for commands followed by “&” are given upon receipt of the first command subsequent to the termination of the command, or when a *wait* is executed. The following is a list of the abnormal termination messages:

Bus error  
Trace/BPT trap



Illegal instruction  
 IOT trap  
 EMT trap  
 Bad system call  
 Quit  
 Floating exception  
 Memory violation  
 Killed  
 Broken Pipe  
 Alarm clock  
 Terminated

If a core image is produced, “— Core dumped” is appended to the appropriate message.

**Redirection of I/O.** There are three character sequences that cause the immediately following string to be interpreted as a special argument to the Shell itself. Such an argument may appear anywhere among the arguments of a simple command, or before or after a parenthesized command list, and is associated with that command or command list.

An argument of the form “<arg” causes the file “arg” to be used as the standard input (file descriptor 0) of the associated command.

An argument of the form “>arg” causes file “arg” to be used as the standard output (file descriptor 1) for the associated command. “Arg” is created if it did not exist, and in any case is truncated at the outset.

An argument of the form “>>arg” causes file “arg” to be used as the standard output for the associated command. If “arg” did not exist, it is created; if it did exist, the command output is appended to the file.

For example, either of the command lines

```
ls >junk; cat tail >>junk
( ls; cat tail ) >junk
```

creates, on file “junk”, a listing of the working directory, followed immediately by the contents of file “tail”.

Either of the constructs “>arg” or “>>arg” associated with any but the last command of a pipeline is ineffectual, as is “<arg” in any but the first.

In commands called by the Shell, file descriptor 2 refers to the standard output of the Shell regardless of any redirection of standard output. Thus filters may write diagnostics to a location where they have a chance to be seen.

A redirection of the form “<—” requests input from the standard input that existed when the instance of the Shell was created. This permits a command file to be treated as a filter. The procedure “lower” could be used in a pipeline to convert characters to lower case:

```
tr "[A-Z]" "[a-z]" <—
```

A typical invocation might be:

```
reform -8 -c <prnt0 | lower >prnt0a
```

**Generation of argument lists.** If any argument contains any of the characters “?”, “\*” or “[”, it is treated specially as follows. The current directory is searched for files which *match* the given argument.

The character "\*" in an argument matches any string of characters in a file name (including the null string).

The character "?" matches any single non-null character in a file name.

Square brackets "[...]" specify a class of characters which matches any single file name character in the class. Within the brackets, each ordinary character is taken to be a member of the class. A pair of characters separated by "-" places in the class each character lexically greater than or equal to the first and less than or equal to the second member of the pair.

Other characters match only the same character in the file name.

If an argument starts with "\*", "?", or "[", that argument will not match any file name that starts with ".".

For example, "\*" matches all file names; "?" matches all one-character file names; "[ab]\*.s" matches all file names beginning with "a" or "b" and ending with ".s"; "?[zi-m]" matches all two-character file names ending with "z" or the letters "i" through "m". None of these examples match names that start with ".".

If the argument with "\*" or "?" also contains a "/", a slightly different procedure is used: instead of the current directory, the directory used is the one obtained by taking the unmodified argument to the "/" preceding the first "\*?[". The matching process matches the remainder of the argument after this "/" against the files in the derived directory. For example: "/usr/dmr/a\*.s" matches all files in directory "/usr/dmr" which begin with "a" and end with ".s".

In any event, a list of names is obtained which match the argument. This list is sorted into alphabetical order, and the resulting sequence of arguments replaces the single argument containing the "\*", "[", or "?". The same process is carried out for each argument (the resulting lists are *not* merged) and finally the command is called with the resulting list of arguments.

If a command has one argument with "\*", "?", or "[", a diagnostic is printed if no file names match that argument. If a command has several such arguments, a diagnostic is only printed if they *all* fail to match any files.

**Quoting.** The character "\" causes the immediately following character to lose any special meaning it may have to the Shell; in this way "<", ">", and other characters meaningful to the Shell may be passed as part of arguments. A special case of this feature allows the continuation of commands onto more than one line: a new-line preceded by "\" is translated into a blank.

A sequence of characters enclosed in single quotes (') is taken literally, with no substitution or special processing whatsoever.

Sequences of characters enclosed in double quotes (") are also taken literally, except that "\", "", and "\$" are handled specially. The sequences "\" and "\"\$" yield "" and "\$", respectively. The sequence "\x", where "x" is any character except "" or "\$", yields "\x". A "\$" within a quoted string is processed in the same manner as a "\$" that is not in a quoted string (see below), unless it is preceded by a "\". For example:

```
ls | pr -h "\"My directory\"$"
```

causes a directory listing to be produced by *ls*, and passed on to *pr* to be printed with the heading "\"My directory\$". Quotes permit the inclusion of blanks in the heading, which is a single argument to *pr*. Note that "\" inside quotes disappears only when preceding "\$" or "".

**Argument passing.** When the Shell is invoked as a command, it has additional string processing capabilities. Recall that the form in which the Shell is invoked is

```
sh [ -v ] [ name [ arg1 ... ] ]
```

The *name* is the name of a file which is read and interpreted. If not given, this subinstance of the Shell continues to read the standard input file.

In command lines in the file (and also in command input), character sequences of the form “\$N”, where *N* is a digit, are replaced by the *n*th argument to the invocation of the Shell (*argn*). “\$0” is replaced by *name*. Shell variables (“\$a” – “\$z”), described below, are replaced in the same way.

The special argument “\$\*” is a name for the *current* sequence of all arguments from “\$1” through the last argument, each argument separated from the previous by a single blank.

The special argument “\$\$” is the ASCII representation of the unique process number of the current Shell. This string is useful for creating temporary file names within command files.

The sequence “\$x”, where “x” is any character except one of the 38 characters mentioned above, is taken to refer to a variable “x” whose value is the null string. All substitution on a command line occurs *before* the line is interpreted: no action that alters the value of any variable can have any effect on a reference to that variable that occurs on the *same* line.

The argument `-t`, used alone, causes *sh* to read the standard input for a single line, execute it as a command, and then exit. It is useful for interactive programs which allow users to execute system commands.

The argument `-c` (used with one following argument) causes the next argument to be taken as a command line and executed. No new-line need be present, but new-line characters are treated appropriately. This facility is useful as an alternative to `-t` where the caller has already read some of the characters of the command to be executed.

The argument `-v` (“verbose”) causes every command line to be printed after all substitution occurs, but before execution. Each argument is preceded by a single blank. When given, the `-v` must be the first argument.

Used alone, the argument “-” suppresses prompting, and is commonly used when piping commands into the Shell:

```
ls | sed "s/./echo &;cat &/" | sh -
```

prints all files in a directory, each prefaced by its name.

**Initialization.** When the Shell is invoked under the name “-” (as it is when you login), it attempts to read the file “.profile” in the current directory and execute the commands found there. When it finishes with “.profile”, the Shell prompts the user for input as usual. Typical files contain commands to set terminal tabs and modes, initialize values of Shell variables, look at mail, etc.

**End of file.** An end-of-file in the Shell’s input causes it to exit. A side effect of this fact means that the way to log out from UNIX is to type an EOT.

**Command file errors; interrupts.** Any Shell-detected error, or an interrupt signal, during the execution of a command file causes the Shell to cease execution of that file. (Except after *onintr*; see below.)

Processes that are created with “&” ignore interrupts. Also if such a process has not redirected its input with a “<”, its input is automatically redirected to come from the zero length file “/dev/null”.

**Special commands.** The following commands are treated specially by the Shell. These commands generally do not work when named as arguments to programs like *time*, *if*, or *nohup*.

because in these cases they are not invoked directly by the Shell.

*chdir* and *cd* are done without spawning a new process by executing *chdir*(II).

*login* is done by executing “/bin/login” without creating a new process.

*wait* is done without spawning a new process by executing *wait*(II).

*shift* [ *integer* ] is done by manipulating the arguments to the Shell. In the normal case, *shift* has the effect of decrementing the Shell argument names by one (“\$1” disappears, “\$2” becomes “\$1”, etc.). When the optional *integer* is given, only arguments equal to or greater than that number are shifted.

“:” is simply ignored.

“=” *name* [ *arg1* [ *arg2* ] ]

The single character Shell variable (*name*) is assigned a value, either from the optional argument(s), or from standard input. If a single argument is given, its value is used. If a second argument is included, its value is used only if the first argument has a null value. This permits a simple way of setting up default values for arguments:

= a "\$1" default

causing default to be used if “\$1” is null or omitted entirely.

Such variables are referred to later with a “\$” prefix. The variables “\$a” through “\$m” are guaranteed to be initialized to null, and will never have special meanings. The variables “\$n” through “\$z” are *not* guaranteed to be initialized to null, and may, at some time in the future, acquire special meanings. Currently, these variables have predefined meanings:

\$n is the argument count to the Shell command.

\$p contains the Shell directory search sequence for command execution. Alternatives are separated by “:”. The default initial value is:

= p “:/bin:/usr/bin”

which executes from the current directory (the null pathname), then from “/bin”, then from “/usr/bin”, as described above. For the super-user, the value is:

= p “/bin:/etc/”

Using the same syntax, users may choose their own sequence by storing it in a file named “.path” in their login directory. The “.path” information is available to successive Shells; the “\$p” value is not. If the “.path” file contains a second line, it is interpreted as the name of the Shell to be invoked to interpret Shell procedures. (See “\$z” below).

\$r is the exit status code of the preceding command. “0” is the normal return from most commands.

\$s is your login directory.

\$t is your login tty letter.

\$w is your file system name (first component of “\$s”).

\$z is the name of the program to be invoked when a Shell procedure is to be executed. Its default value is “/bin/sh”, but it can be overridden by supplying a second line in the “.path” file. It can be used to achieve consistent use of a specific Shell during periods when several distinct Shells are present in the system. For safety in the presence of change, use “\$z” as a command rather than “sh”.

No substitution of variables (or arguments) occurs within single quotes ('). Within double quotes ("), a variable string is substituted unchanged, even if it contains characters (“”, “\”, or “\$”) that might otherwise be treated specially. In particular, the argument “\$1” can be passed

unchanged to another command by using “\$1”. Outside quotes, substituted characters possess the same special meanings they have as if typed directly.

To illustrate, suppose that the shell procedure “mine” is called with two arguments:

```
sh mine 'a; echo "$2" ""
```

Then sample commands in “mine” and their output are as follows:

```
echo '$1'           $1
echo "$1"          a; echo "$2"
echo $1            a
                  $2
echo $2a"         a
echo "$2a"        "a
echo $2           syntax error
```

The appearance of the string “\$2” (rather than “”) occurs because the Shell performs only one level of substitution, i.e., no rescanning is done.

*onintr* [ *label* ]

Causes control to pass to the label named (using a *goto* command) if the Shell command file is interrupted. After such a transfer, interrupts are re-enabled. *Onintr* without an argument also enables interrupts. The special label “-” will cause any number of interrupts to be ignored.

*next* [ *name* ]

This command causes *name* to become the standard input. Current input is never effectively resumed. If the argument is omitted, your terminal keyboard is assumed.

*pump* [ -[*subchar*] ] [ + ] [ *eofstr* ]

This command reads its standard input until it finds *eofstr* (defaults to “!” if not specified) alone on a line. It normally substitutes the values of arguments and variables (marked with “\$” as usual). If “-” is given alone, substitution is suppressed, and “-*subchar*” causes *subchar* to be used in place of “\$” as the indicator character for substitution. Escaping is handled as in quoted strings: the indicator character may be escaped by preceding it by “\”. Otherwise, “\” and other characters are transmitted unchanged. If “+” is used, leading tabs in the input are thrown away, allowing indentation. This command may be used interactively and in pipelines.

*opt* [ -v ] [ +v ] [ -p *prompt-str* ]

The argument -v turns on tracing, in the same style as a -v argument for the Shell. The argument +v turns it off. The argument -p causes the next argument string to be used as the prompt string for an interactive shell.

*Commands implementing control structure.* Control structure is provided by a set of commands that happen currently to be built into the Shell, although no guarantee is given that this will remain so. They are documented separately as follows:

```
if(I) - if, else, endif, and test.
switch(I) - switch, breaksw, endsw.
while(I) - while, end, break, continue.
goto(I) - goto.
exit(I) - exit.
```

## FILES

```
/etc/sha, for shell accounting.
/dev/null as a source of end-of-file.
.path in login directory to initialize $p and name of Shell.
.profile in login directory for general initialization.
```

**SEE ALSO**

*The UNIX Time-Sharing System* by D. M. Ritchie and K. Thompson, CACM, July, 1974, which gives the theory of operation of the Shell.

*PWB/UNIX Shell Tutorial* by J. R. Mashey.

chdir(I), equals(I), exit(I), expr(I), fd2(I), if(I), login(I), loginfo(I), onintr(I), pump(I), shift(I), switch(I), wait(I), while(I), pexec(III), sha(V), glob(VIII)

**EXIT CODE**

If an error occurs in a command file, the Shell returns the exit value "1" to the parent process. Otherwise, the current value of the Shell variable *\$r* is returned. Execution of a command file is terminated by an error.

**BUGS**

There is no built-in way to redirect the diagnostic output; *fd2(I)* must be used.

A single command line is limited to 1000 total characters, 50 arguments, and approximately 20 operators.

**NAME**

shift — adjust Shell arguments

**SYNOPSIS**

shift [ digit ]

**DESCRIPTION**

*Shift* is used in Shell command files to shift the argument list left by 1, so that old **\$2** can now be referred to by **\$1** and so forth. *Shift* is useful to iterate over several arguments to a command file. For example, the command file

```
while "$1"
    pr -3 $1
    shift
end
```

prints each of its arguments in 3-column format.

*Shift* is executed within the Shell.

The optional argument causes *shift* to leave shell arguments numbered lower than *\$digit* alone on shifts; *shift* alone and *shift 1* are identical in effect.

**SEE ALSO**

sh(1)

**NAME**

size — size of an object file

**SYNOPSIS**

size [ object ... ]

**DESCRIPTION**

*Size* prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. If no file is specified, **a.out** is used.

**SEE ALSO**

a.out(V)



**NAME**

sleep – suspend execution for an interval

**SYNOPSIS**

**sleep** time

**DESCRIPTION**

*Sleep* suspends execution for *time* seconds. It is used to execute a command in a certain amount of time as in:

```
(sleep 105; command)&
```

Or to execute a command every so often, as in this shell command file:

```
while 1
    command
    sleep 37
end
```

**SEE ALSO**

sleep(1)

**BUGS**

*Time* must be less than 65536 seconds.

**NAME**

sno – Snobol interpreter

**SYNOPSIS**

sno [ file ]

**DESCRIPTION**

*Sno* is a Snobol III (with slight differences) compiler and interpreter. *Sno* obtains input from the concatenation of *file* and the standard input. All input through a statement containing the label 'end' is considered program and is compiled. The rest is available to 'syspit'.

*Sno* differs from Snobol III in the following ways.

There are no unanchored searches. To get the same effect:

```
a ** b           unanchored search for b
a *x* b = x c    unanchored assignment
```

There is no back referencing.

```
x = "abc"
a *x* x          is an unanchored search for 'abc'
```

Function declaration is different. The function declaration is done at compile time by the use of the label 'define'. Thus there is no ability to define functions at run time and the use of the name 'define' is preempted. There is also no provision for automatic variables other than the parameters. For example:

```
define f()
```

or

```
define f(a,b,c)
```

All labels except 'define' (even 'end') must have a non-empty statement.

If 'start' is a label in the program, program execution will start there. If not, execution begins with the first executable statement. 'define' is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators '/' and '\*' must be set off by space.

The right side of assignments must be non-empty.

Either '' or "" may be used for literal quotes.

The pseudo-variable 'syspt' is not available.

**SEE ALSO**

Snobol III Manual (JACM Vol. 11, No. 1; Jan. 1964; pp. 21ff.)

## NAME

sort – sort or merge files

## SYNOPSIS

sort [ **-mubdfnr** ] [ **-tx** ] [ **+pos** [ **-pos** ] ] ... [ **-o name** ] [ **name** ] ...

## DESCRIPTION

*Sort* sorts lines of all the named files together and writes the result on the standard output. The name ‘–’ means the standard input. The standard input is also used if no input file names are given. Thus *sort* may be used as a filter.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected by the following flags one or more of which may appear.

- b** Leading blanks (spaces and tabs) are not included in keys.
- d** ‘Dictionary’ order: only letters, digits and blanks are significant in comparisons.
- f** Fold lower case letters onto upper case.
- i** Ignore all nonprinting nonblank characters in nonnumeric comparisons.
- n** An initial numeric string, consisting of optional minus sign, digits and optionally included decimal point, is sorted by arithmetic value.
- r** Reverse the sense of comparisons.
- tx** Tab character between fields is *x*.

Selected parts of the line, specified by **+pos** and **-pos**, may be used as sort keys. *Pos* has the form *m.n* optionally followed by one or more of the flags **bdfnr**, where *m* specifies a number of fields to skip, *n* a number of characters to skip further into the next field, and the flags specify a special ordering rule for the key. A missing *.n* is taken to be 0. **+pos** denotes the beginning of the key; **-pos** denotes the first position after the key (end of line by default). Later keys are compared only when all earlier keys compare equal.

When no tab character has been specified, a field consists of nonblanks and any preceding blanks. Under the **-b** flag, leading blanks are excluded from a field. When a tab character has been specified, fields are strings separated by tab characters.

Lines that otherwise compare equal are ordered with all bytes significant.

These flag arguments are also understood:

- m** Merge only, the input files are already sorted.
- o** The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs, except under the merge flag **-m**.
- u** Suppress all but one in each set of contiguous equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

Examples. Print a list of all the distinct *roff*(1) commands in a given document:

```
grep "\." document | sort -u +0 -0.3
```

Print the password file *passwd*(V) sorted by user id:

```
sort -t: +2n /etc/passwd
```

## FILES

/tmp/stm???

**NAME**

spell — find spelling errors

**SYNOPSIS**

spell [ -v ] [ -1 ] file ...

**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, 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* omits *nroff*(1), *troff*(1), *neqn*(1), and *eqn*(1) constructions from the input.

The process may take several minutes.

Under the *-v* flag, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

The *-1* option causes the final output to appear in a single column instead of three columns. The normal header and pagination is also suppressed.

The spelling list is based primarily on Kucera and Francis, *Computational Analysis of Present-Day English* and the Merriam Webster *New International Dictionary, 2nd edition*. Other sources include lists of chemical elements, states, countries, provinces, capital cities, major cities; given names from Kucera and Francis; the most common surnames from a large telephone book; common names from the index of *Fieldbook of Natural History* by E. L. Palmer and H. S. Fowler; selected names from *Bulfinch's Mythology*; Bell System Practices; Bell Laboratories technical papers and manuals; the *Federalist* papers; random literary fragments; etc.

If the file “/usr/dict/spellhist” is writable, *spell* accumulates copies of its output there.

**FILES**

/bin/deroff, /usr/lib/spell[0123]: programs

/usr/lib/w2006: list of common words for primary filtering

/usr/dict/spellinglist

/usr/dict/stoplist: likely misspellings (e.g. thier=thy-y+ier) that would otherwise pass

/usr/dict/spellhist

**SEE ALSO**

typo(1)

**BUGS**

The coverage of the spelling list is uneven; new installations will probably wish to monitor the output for a few months to gather local additions.

## NAME

spline – interpolate smooth curve

## SYNOPSIS

**spline** [ option ] ...

## DESCRIPTION

*Spline* takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *plot*(1).

The following options are recognized, each as a separate argument.

- a** Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.
- k** The next argument is used as the constant  $k$  used in the boundary value computation
$$y_0'' = ky_1'', \quad y_n'' = ky_{n-1}''$$
is set by the next argument. By default  $k = 0$ .
- n** Space output points so that approximately  $n$  points occur between the lower and upper  $x$  limits, where  $n$  is the next argument. (Default  $n = 100$ .)
- p** Make output periodic, i.e. match derivatives at ends. First and last input values should normally agree.
- x** Next 1 (or 2) arguments are lower (and upper)  $x$  limits. Normally these limits are calculated from the data. Automatic abscissas start at the lower limit (default 0).

## SEE ALSO

*plot*(1)

## BUGS

A limit of 1000 input points is enforced silently.

**NAME**

**split** - split a file into pieces

**SYNOPSIS**

**split** -n [ file [ name ] ]

**DESCRIPTION**

*Split* reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if **-** is given in its stead, then the standard input file is used.

**NAME**

**strip** — remove symbols and relocation bits

**SYNOPSIS**

**strip** name ...

**DESCRIPTION**

*Strip* removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **-s** option of *ld*.

**FILES**

/tmp/stm?      temporary file

**SEE ALSO**

ld(1), as(1), nm(1)

## NAME

stty — set terminal options

## SYNOPSIS

stty [ option ... ]

## DESCRIPTION

*Stty* sets certain I/O options on the current output terminal. With no argument, it reports the current settings of the options. The option strings are selected from the following set:

**even** allow even parity  
**—even** disallow even parity  
**odd** allow odd parity  
**—odd** disallow odd parity  
**raw** raw mode input (no erase, kill, interrupt, quit, EOT; parity bit passed back)  
**—raw** negate raw mode  
**cooked** same as '—raw'  
**—nl** allow carriage return for new-line, and output CR-LF for carriage return or new-line  
**nl** accept only new-line to end lines  
**echo** echo back every character typed  
**—echo** do not echo characters  
**lcase** map upper case to lower case  
**—lcase** do not map case  
**—tabs** replace tabs by spaces when printing  
**tabs** preserve tabs  
**ek** reset erase and kill characters back to normal # and @.  
**erase c** set erase character to *c*.  
**kill c** set kill character to *c*.  
**cr0 cr1 cr2 cr3**  
select style of delay for carriage return (see *stty*(II))  
**nl0 nl1 nl2 nl3**  
select style of delay for linefeed (see *stty*(II))  
**tab0 tab1 tab2 tab3**  
select style of delay for tab (see *stty*(II))  
**ff0 ff1**  
select style of delay for form feed (see *stty*(II))  
**tty33** set all modes suitable for the Model 33 TELETYPE®  
**tty37** set all modes suitable for the Model 37 TELETYPE  
**vt05** set all modes suitable for Digital Equipment Corp. VT05 terminal  
**tn300** set all modes suitable for a General Electric TermiNet 300  
**ti700** set all modes suitable for Texas Instruments 700 series terminal  
**tek** set all modes suitable for Tektronix 4014 terminal  
**hup** hang up dataphone on last close.  
**—hup** do not hang up dataphone on last close.  
**0** hang up phone line immediately  
**50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb**  
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

## SEE ALSO

*stty*(II)



**NAME**

`su` — become privileged user

**SYNOPSIS**

`su` [ *name* ]

**DESCRIPTION**

*Su* allows one to become the super-user, who has all sorts of marvelous (and correspondingly dangerous) powers. In order for *su* to do its magic, the user must supply a password. If the password is correct, *su* will execute the Shell with the user-id set to that of the super-user. To restore normal user-id privileges, type an end-of-file to the super-user Shell.

The password demanded is that of the entry “root” in the system’s password file.

To remind the super-user of his responsibilities, the Shell substitutes “#” for its usual prompt (“%”). The ordinary user’s command path search sequence does not apply to the super-user. The super-user gets “/bin”, “/etc”, and “/” instead (no current directory).

The optional argument allows logging in as *name* without logging off as yourself. That is, you get the powers and privileges, if any, of the user whose *login* name is *name*. In this case (unless you already are the super-user), *su* asks for that user’s password, rather than for the super-user password.

**FILES**

/etc/passwd                      system’s password file

**SEE ALSO**

sh(I), pexec(III)

**BUGS**

If you are the super-user and invoke *su* with an invalid argument (i.e., a non-existent *login name*), you will get the powers and privileges, if any, of the user whose user-id is 255 (regardless of whether there actually is such a user).

**NAME**

**sum** — print checksum of a file

**SYNOPSIS**

**sum** [ file ] ...

**DESCRIPTION**

*Sum* sums the contents of the bytes (mod  $2^{16}$ ) of each *file* specified. *Sum* prints the file name, the number of whole or partial 512-byte disk blocks read, and the summed value of its bytes in decimal.

In practice, *sum* is often used to verify that all of a special file can be read without error.

**NAME**

switch — shell multi-way branch command

**SYNOPSIS**

```
switch arg
: label1
    commands...
    breaksw
. . .
: labeln
    commands...
    breaksw
: default
    commands...
endsw
```

**DESCRIPTION**

*Switch* searches forward in the input file for the first one of:

1. a label that pattern-matches *arg*. The pattern-matching used is that of the Shell in generating argument lists.
2. the label *default*.
3. a matching *endsw* command.

The Shell resumes reading commands from the next line after the location where the search stopped. Thus, *switch* supplies a 'case' or 'computed goto' statement similar to that of C. Because ':' is ignored by the Shell, several labels may occur in order, so that the same sequence of commands is executed for several different values of *arg*.

The *breaksw* command searches forward to the next unmatched *endsw*, and is normally used at the end of the sequence of commands following each label. It may be omitted to allow common code to be shared among label values. Several *breaksw* commands may be written on the same line to exit from that many levels of nested *switch-endsw* pairs.

The optional label *default* should be placed last, since *switch* always stops upon discovering it. The construct can be nested: any labels enclosed by a *switch-endsw* pair are ignored by an outer *switch*. The most common use of *switch* is to process 'flag' arguments in a shell procedure.

**SEE ALSO**

if(I), sh(I), while(I)

**DIAGNOSTICS**

switch: missing endsw  
breaksw: missing endsw

**BUGS**

None of these commands should be hidden behind semicolons. Nested groups hidden behind *if* or *else* may also cause trouble.

**NAME**

`sync` — update the super block

**SYNOPSIS**

`sync`

**DESCRIPTION**

*Sync* executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. See *sync(II)* for details.

**SEE ALSO**

*sync(II)*

## NAME

`tabs` — set tabs on terminal

## SYNOPSIS

`tabs` [`tabspec`] [`+f`] [`+mn`] [`+ln`] [`+ttype`] [`+q`]

## DESCRIPTION

`Tabs` sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs, including the DASI450 (DIABLO 1620 or XEROX 1700), GSI300 (DTC300 or DASI300), DASI300S (DTC300S), HP2640B (HP2640A, HP2644A, HP2645A, etc.), TELETYPE® Model 40/2, and General Electric TermiNet terminals.

Users of TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings; the first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left by a TermiNet, but not by other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is also possible to set a left margin on the DASI450 and DASI300S, although in a different way.

Four types of tab specification are accepted for *tabspec*: 'canned', repetitive, arbitrary, and file. If no arguments are given, the default value is `-8`, i.e., UNIX 'standard' tabs. The lowest column number is 1 and the highest is 158. 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 DASI300, DASI300S, and DASI450.

`-code` Gives the name of one of a set of 'canned' 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. In order to get *send(1)* to prepend the blanks at the beginning, files using this tab setup should include a format specification (see *fspec(V)*) as follows:  
<:t-c2 m6 s66 d:>

`-c3` 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67  
COBOL compact format (columns 1-6 omitted), with more tabs than `-c2`. THIS IS THE RECOMMENDED FORMAT FOR COBOL. The appropriate format specification is:  
<:t-c3 m6 s66 d:>

- f 1,7,11,15,19,23  
FORTRAN
- p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61  
PL/I
- s 1,10,55  
SNOBOL
- u 1,12,20,44  
UNIVAC 1100 Assembler

In addition to these 'canned' formats, three other types exist:

- n A repetitive specification requests tabs at columns  $1+n$ ,  $1+2*n$ , etc. Note that such a setting leaves a left margin of  $n$  columns on TermiNet terminals *only*. Of particular importance is the value  $-8$ : this represents the UNIX 'standard' tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff(1)*  $-h$  option for high-speed output (about 10% speed increase). 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. The maximum tab value accepted is 158. 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 tab lists 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(V)*). If it finds one there, it sets the tab stops according to it, otherwise it sets them as  $-8$ . If an actual format specification is found in the file, it is printed at the terminal to remind the user what it is, unless the  $+f$  flag is also included to suppress this output. 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(1)* command:

**tabs** --file; pr file

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect.

- +f If the --file type of tab specification is used and this option given, no tab specification is printed at the terminal, but the tab stops are set. This option is most useful when *tabs* is invoked from a shell procedure or another command, rather than directly from a terminal. This option has no effect unless a --file form of specification is used.
- +ln The length argument gives the number of the rightmost column at which a tab will be set by a repetitive-style specification. The default value is 132, but may be different if the  $+t$  argument implies a more appropriate value for the specific type of terminal being used. When examining printed output obtained from another computer, it is helpful to have tabs across the entire width of the terminal. Maximum usable values of  $n$  are 118 (TermiNet), 132 (any DASI in 10-pitch mode), 158 (any DASI in 12-pitch mode), and 80 (HP2640). Although *tabs* will accept larger values without diagnostics, using them may cause a terminal (especially a DASI) to behave strangely.
- +mn The margin argument may be used for TermiNet, DASI450, and DASI300S 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.

To reset the left margin of a DASI450 to the normal(leftmost) position, type:

```
tabs +t450 +m0
```

The margin on a DASI450 or DASI300S is reset only when the **+m** flag is given explicitly. The margin is not settable on a DASI300, and is settable on the DASI450 and DASI300S *only* when the **+t** option is used (see below).

**+q** The ('quick') flag suppresses the emission of characters to clear previously set tabs. It can be used if the terminal is known to be clear already, i.e., just after it has been powered up or reset.

**+ttype** The terminal type can be supplied to help *tabs* optimize its output for specific kinds of terminals, and is sometimes required when certain functions of some terminals are desired. This argument interacts with **+l** by setting different defaults for different terminals, and different maximum lengths. It interacts with **+m** because different methods must be used to set margins on the various terminals.

Given below are the possible cases for **+t** argument, listing argument value, maximum length, default length if **+l** is omitted, and notes. The notes give the following codes: 'S' for a short (several characters) clearing sequence, 'L' for a long sequence (about 60 characters), 'M' for a settable margin, and a list of the terminal types expected.

Value	Maximum	Default	S/L	M	Terminal(s)
<b>+t300</b>	158	132	L		GSI300 (DTC300 or DASI300)
<b>+tgsi</b>	"	"	"		"
<b>+t300S</b>	158	132	S	M	DASI300S (DTC300S)
<b>+tgsis</b>	"	"	"	"	"
<b>+t450</b>	158	132	S	M	DASI450 (DIABLO 1620 or XEROX 1700)
<b>+t1620</b>	"	"	"	"	"
<b>+ttn</b>	118	118	S	M	TermiNet 300 or 1200
<b>+thp</b>	80	80	L		HP2640A, HP2640B
<b>+t40-2</b>	80	80	S		TELETYPE 40/2
<b>+t</b>	158	132	S		any with settable tabs
omitted	158	132	L		any with settable tabs

Omitting the **+t** argument entirely will work for most situations. You should probably try to type the least that will work, and be more specific only when necessary.

Tab-setting is performed using the standard output.

#### DIAGNOSTICS

"illegal tabs" when arbitrary tabs are ordered incorrectly, or include any value greater than 158.

"illegal increment" when a zero or missing increment value is found in an arbitrary specification.

"unknown tab code" when a 'canned' code cannot be found.

"can't open" if **---file** option used, and file can't be opened.

"file indirection" if **---file** option used and the specification in that file points to yet another file.

Indirection of this form is not permitted.

**EXIT CODES**

- 0 - normal
- 1 - for any error

**SEE ALSO**

fspec(V), nroff(I), reform(I), send(I)  
GSI300(VII), DASI450(VII), HP2640(VII), TERMINET(VII)

**BUGS**

It is sad, but true, that it is often necessary to specify the terminal type. Various terminals use totally inconsistent ways of clearing tabs and setting margins. *Tabs* clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.



**NAME**

**tail** – deliver the last part of a file

**SYNOPSIS**

**tail** [ ±number[lbc] ] [ file ]

**DESCRIPTION**

*Tail* copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *-number* from the end of the input. *Number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b** or **c**. When no units are specified, counting is by lines.

**SEE ALSO**

dd(I)

**BUGS**

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

## NAME

`tbl` -- format tables for `nroff` or `troff`

## SYNOPSIS

`tbl` [ files ] ...

## DESCRIPTION

*Tbl* is an *nroff*(1) or *troff*(1) preprocessor for formatting tables. The input files are copied to the standard output, except for lines between `.TS` and `.TE` command lines, which are assumed to describe tables, and which are reformatted. There are several global options; if any are desired, they are specified on the first line after `.TS` as a series of keywords separated by blanks or commas and followed by a semicolon. The possible words are:

- center - center the table
- expand - format the table to fill the current line length
- box - enclose the table in a box
- allbox - draw all possible lines so that every item is in a box

After this line (or after `.TS` if no global options are given) are the lines describing the table format. Each line describes a line of the actual table. One letter is used for each column. As many lines as needed to describe the table are given; the last line should end with the character “.” to signal the end of the format information. The last line of the description will apply to all following lines of the table. The legal characters to describe a column are:

- c center within the column
- r right-adjust
- l left-adjust
- n numerical adjustment: the units digits of numbers are aligned.
- s span the previous entry over this column.
- \_ replace this entry with a horizontal line
- = replace this entry with a double horizontal line

A column letter may be followed by an integer giving the number of spaces between this column and the next; 3 is default. A column letter may be preceded by a “|” character to indicate that a vertical line is to be drawn to the left of this column. Letting `\t` represent a tab (which must be typed as a genuine tab), the input:

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields:

Town	Household Population	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Branchburg	1644	3.49
Bridgewater	7897	3.81
Far Hills	240	3.19

If a table element contains only “\_” or “=”, a single or double line (respectively) is drawn across the *column* at that point. If a table line contains only “\_” or “=”, a single or double line (respectively) is drawn all the way across the *table*.

If a column describer contains the character “|”, a vertical line is drawn to the left of that column beginning at the point in the column corresponding to the position of the vertical bar in the describer, and extending to the bottom of the table.

If no arguments are given, *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn* or *neqn*, the *tbl* command should be first, to minimize the volume of data passed through pipes.

**SEE ALSO**

*TBL* — *A Program to Format Tables* by M. E. Lesk.

**NAME**

tee — pipe fitting

**SYNOPSIS**

tee [ name ... ]

**DESCRIPTION**

*Tee* transcribes the standard input to the standard output and makes copies in the named files.

**NAME**

`time` - time a command

**SYNOPSIS**

`time` command

**DESCRIPTION**

The given command is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command.

The times are printed on the diagnostic output stream.

**BUGS**

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

**NAME**

*tp* — manipulate DECTape and magtape

**SYNOPSIS**

*tp* [ key ] [ name ... ]

**DESCRIPTION**

*tp* saves and restores files on DECTape or magtape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

- r** The named files are written on the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so './abc' can never be the same as '/usr/dmr/abc' even if '/usr/dmr' is the current directory. If no file argument is given, '.' is the default.
- u** updates the tape. **U** is like **r**, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. **U** is the default command if none is given.
- d** deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.
- x** extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.
- t** lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

- m** Specifies magtape as opposed to DECTape.
- 0,...,7** This modifier selects the drive on which the tape is mounted. For DECTape, 'x' is default; for magtape '0' is the default.
- v** Normally *tp* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.
- c** means a fresh dump is being created; the tape directory is zeroed before beginning. Usable only with **r** and **u**. This option is assumed with magtape since it is impossible to selectively overwrite magtape.
- f** causes new entries on tape to be 'fake' in that no data is present for these entries. Such fake entries cannot be extracted. Usable only with **r** and **u**.
- i** Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.
- w** causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response **x** means 'exit'; the *tp* command terminates immediately. In the **x**

function, files previously asked about have been extracted already. With **r**, **u**, and **d** no change has been made to the tape.

**FILES**

/dev/tap?

/dev/mt?

**DIAGNOSTICS**

Several; the non-obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

**BUGS**

A single file with several links to it is treated like several files.

## NAME

tr - transliterate

## SYNOPSIS

tr [ **-cds** ] [ string1 [ string2 ] ]

## DESCRIPTION

*Tr* copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **-cds** may be used: **-c** complements the set of characters in *string1* with respect to the universe of characters whose ascii codes are 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* - *b*] stands for the string of characters whose ascii codes run from character *a* to character *b*.

[*a*\**n*], where *n* is an integer or empty, stands for *n*-fold repetition of character *a*. *N* is taken to be octal or decimal according as its first digit is or is not zero. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character '\ ' may be used as in the Shell to remove special meaning from any character in a string. In addition, '\ ' 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 alphabetic. 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

sh(I), ed(I), ascii(V)

## BUGS

Won't handle ascii NUL in *string1* or *string2*; always deletes NUL from input.



## NAME

nroff, troff — text formatters

## SYNOPSIS

**nroff** (or **troff**) [ options ] files

## DESCRIPTION

NROFF and TROFF accept lines of text interspersed with lines of format control information and format the text into a printable, paginated document having a user-designed style. NROFF and TROFF are highly compatible with each other and it is almost always possible to prepare input acceptable to both. Conditional input is provided that enables the user to embed input expressly destined for either program. NROFF can prepare output directly for a variety of terminal types and is capable of utilizing the full resolution of each terminal.

An argument consisting of a single minus (—) is taken to be a file name corresponding to the standard input. If no file names are given, input is taken from the standard input. The options, which may appear in any order so long as they appear before the filenames, are:

<i>Option</i>	<i>Effect</i>
<b>-olist</b>	Print only pages whose page numbers appear in <i>list</i> , which consists of numbers and number ranges separated by commas. A number range has the form <i>N-M</i> and means pages <i>N</i> through <i>M</i> inclusive; an initial <i>-N</i> means from the beginning to page <i>N</i> ; and a final <i>N-</i> means from <i>N</i> to the end.
<b>-nN</b>	Number first generated page <i>N</i> .
<b>-sN</b>	Stop every <i>N</i> pages. NROFF will halt prior to every <i>N</i> pages (default <i>N=1</i> ) to allow paper loading or changing, and will resume upon receipt of a new-line character. TROFF will stop the phototypesetter every <i>N</i> pages, produce a trailer to allow the changing of cassettes, and will resume after the phototypesetter START button is pressed.
<b>-mname</b>	Prepends the macro file <i>/usr/lib/tmac.name</i> to the input files.
<b>-raN</b>	Register <i>a</i> (one-character name) is set to <i>N</i> .
<b>-i</b>	Read standard input after the input files are exhausted.
<b>-q</b>	Invoke the simultaneous input-output mode of the <b>rd</b> request.

*NROFF Only*

<b>-Ttype</b>	Specifies the output terminal type. Currently defined values for <i>type</i> are <b>37</b> for the (default) Model 37 TELETYPE®, <b>tn300</b> for the GE TermiNet 300 (or any terminal without half-line capabilities), <b>300</b> for the DASI-300, <b>450</b> for the DASI-450 (or Diablo Hyterm) and <b>300S</b> for the DASI-300S. For 12-pitch, use <b>300-12</b> , <b>300S-12</b> , and <b>450-12</b> .
<b>-e</b>	Produce equally-spaced words in adjusted lines, using full terminal resolution.
<b>-h</b>	Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

*TROFF Only*

<b>-t</b>	Direct output to the standard output instead of the phototypesetter.
<b>-f</b>	Refrain from feeding out paper and stopping phototypesetter at the end of the run.

- w** Wait until phototypesetter is available, if it is currently busy.
- b** TROFF will report whether the phototypesetter is busy or available. No text processing is done.
- a** Send a printable (ASCII) approximation of the results to the standard output.
- pN** Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
- g** Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output.

**FILES**

/usr/lib/suftab suffix hyphenation tables  
/tmp/ta00000 temporary file  
/usr/lib/tmac.\* standard macro files  
/usr/lib/term/\* (NROFF only) terminal driving tables  
/usr/lib/font/\* (TROFF only) font width tables

**SEE ALSO**

*NROFF/TROFF User's Manual* by J. F. Ossanna.  
*A TROFF Tutorial* by B. W. Kernighan.  
tbl(1).  
For NROFF, see neqn(1), col(1), and tabs(1)  
For TROFF, see eqn(1).

**NAME**

tty – get terminal name

**SYNOPSIS**

tty

**DESCRIPTION**

*Tty* gives the name of the user's terminal in the form 'ttn' for *n* a digit or letter. The actual path name is then '/dev/ttn'.

**DIAGNOSTICS**

'not a tty' if the standard input file is not a terminal.

**NAME**

typo — find possible typos

**SYNOPSIS**

typo [ **-1** ] [ **-n** ] file ...

**DESCRIPTION**

*Typo* hunts through a document for unusual words, typographic errors, and *hapax legomena* and prints them on the standard output.

The words used in the document are printed out in decreasing order of peculiarity along with an index of peculiarity. An index of 10 or more is considered peculiar. Printing of certain very common English words is suppressed.

The statistics for judging words are taken from the document itself, with some help from known statistics of English. The **-n** option suppresses the help from English and should be used if the document is written in, for example, Urdu.

The **-1** option causes the final output to appear in a single column instead of three columns. The normal header and pagination is also suppressed.

*Roff*(1) and *nroff*(1) control lines are ignored. Upper case is mapped into lower case. Quote marks, vertical bars, hyphens, and ampersands within words are equivalent to spaces. Words hyphenated across lines are put back together.

**FILES**

/tmp/ttmp??  
/usr/lib/salt  
/usr/lib/w2006

**SEE ALSO**

spell(1)

**BUGS**

Because of the mapping into lower case and the stripping of special characters, words may be hard to locate in the original text.

The escape sequences of *troff*(1) are not correctly recognized.

**SEE ALSO**

spell(1)

**NAME**

uname — print name of current UNIX

**SYNOPSIS**

**uname**

**DESCRIPTION**

*Uname* prints the current name of UNIX on the standard output file. It is mainly useful to determine what system one is using.

**SEE ALSO**

uname(II)

**NAME**

**uniq** — report repeated lines in a file

**SYNOPSIS**

**uniq** [ **-udc** [ **+n** ] [ **-n** ] ] [ input [ output ] ]

**DESCRIPTION**

*Uniq* reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see *sort(1)*. If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

- n** The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.
- +n** The first *n* characters are ignored. Fields are skipped before characters.

**SEE ALSO**

*sort(1)*, *comm(1)*

**NAME**

units – conversion program

**SYNOPSIS**

units

**DESCRIPTION**

*Units* converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```

You have: inch
You want: cm
          * 2.54000e+00
          / 3.93701e-01

```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```

You have: 15 pounds force/in2
You want: atm
          * 1.02069e+00
          / 9.79730e-01

```

*Units* only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. For a complete list of units, 'cat /usr/lib/units'.

**FILES**

/usr/lib/units

## NAME

vp — Versatec print

## SYNOPSIS

vp [-bbin] [-ooffset] [-ttspec] [-n] [-rname] cmd [args]

## DESCRIPTION

*vp* builds a *sh*(1) command file in directory */usr/vpd*, and invokes */etc/vpd* (the Versatec daemon). The command file has the form:

```
: logname
vpbrk bin logname
chdir curdir
= p < logdir/.path
cmd args ^ reform tspec -0 +poffset
cat /usr/vpd/.X
[ echo shfile finished ^ mail logname >/dev/null ]
[ echo shfile finished ^ write logname >/dev/null ]
rm -f shfile
```

Here *logname* is your login name, *curdir* is your current directory when you executed *vp*, *logdir* is your login directory, *shfile* is the name that *vp* selects for the generated command file, *bin* is your data station bin (see below), *offset* is the offset for *reform*(1) (see below), and *cmd* and *args* are the command and optional arguments specified on the command line.

The Versatec daemon, */etc/vpd*, invokes *sh*(1) on the command files that *vp* queues up in */usr/vpd*. The daemon redirects the standard output of each command file to the Versatec printer.

The keyletter arguments are as follows:

- b Your data station bin.
- o The offset for *reform*. The default is 12.
- t The first tabspec for *reform*. The default is '—'.
- n A flag that includes the optional "mail" and "write" lines in the command file.
- r The file named *name* is to be removed after printing is completed.

Example:

```
vp -bx123 pr -l84 myfile
```

## OPERATIONS NOTE:

Execute */etc/vpd* after replacing paper in the Versatec printer.

## FILES

<i>/usr/vpd/*</i>	queued command files
<i>/usr/vpd/.X</i>	terminator
<i>/usr/bin/vpbrk</i>	break page generator
<i>/dev/vp0</i>	Versatec printer
<i>/etc/vpd</i>	daemon program

## BUGS

There should be a *vp*(IV) and a *vpd*(VIII).

Only printers with DMA interfaces are handled; plotting is tolerated, but not supported.

You cannot pipe into *vp*.



**NAME**

*wait* — await completion of process

**SYNOPSIS**

***wait***

**DESCRIPTION**

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the *wait(II)* system call must be executed in the parent process, the Shell itself executes *wait*, without creating a new process.

**SEE ALSO**

*sh(I)*

**BUGS**

After executing *wait* you are committed to waiting until termination, because interrupts and quits are ignored by all processes concerned. The only out, if the process does not terminate, is to *kill* it (see *kill (I)*) from another terminal or to hang up.

**NAME**

`wc` — word count

**SYNOPSIS**

`wc` [ `-l` ] [ name ... ]

**DESCRIPTION**

`Wc` counts lines and words in the named files, or in the standard input if no name appears. A word is a maximal string of printing characters delimited by spaces, tabs or newlines. All other characters are simply ignored.

The `-l` flag suppresses all output except the line count.

**NAME**

what — identify files

**SYNOPSIS**

**what** name ...

**DESCRIPTION**

*What* searches the given files for all occurrences of the pattern which *get(I)* 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 iden____[] "@(#)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 SCCS command *get(I)*, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

**SEE ALSO**

*get(I)*, *help(I)*

**DIAGNOSTICS**

Use *help(I)* for explanations.

**BUGS**

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

**NAME**

whatsnew – compare file modification dates

**SYNOPSIS**

**whatsnew** [ *-yymmdd* ] [ *listfile* ]

**DESCRIPTION**

*Whatsnew* will compare the modification dates of files listed in *listfile* against the date supplied and report those files that have changed since that date. By default, the modification time of the file *.newdate* in the user's login directory will be used as the comparison date. Similarly, the file *.newlist* in the user's login directory will be used if *listfile* is omitted.

If a date is not supplied and *.newlist* exists, it will be re-created. This will essentially update the default comparison date used by subsequent *whatsnew* commands.

Entries in the list file should be relative to the login directory, one per line. If an entry is a directory, files in that directory will be compared. Only one level of directory searching is performed.

**FILES**

/logindir/.newlist  
/logindir/.newdate

**DIAGNOSTICS**

“bad date” if the supplied date is earlier than 1970.

“cannot read list” if the list file is not readable.

“cannot access file status” if it can't.

## NAME

while – shell iteration command

## SYNOPSIS

```
while expr
    commands... (may include break or continue)
end
```

## DESCRIPTION

*While* evaluates the expression *expr*, which is similar to (and a superset of) the expression described in *if(1)*. If the expression is true, *while* does nothing, permitting the command(s) on following lines to be read and executed by the Shell. If the expression is false, the input file is effectively searched for the matching *end* command, and the Shell resumes execution of the command(s) on the line following the *end*. The *while-end* grouping may be nested up to three levels deep.

In addition to the type of expression permitted by *if*, *while* treats a single, nonnull argument as a true expression, and treats a single null argument or lack of arguments as a false expression.

The *break* command terminates the nearest enclosing *while-end* group, causing execution to resume after the nearest succeeding unmatched *end*. Exit from *n* levels is obtained by writing *n break* commands on the same line.

The *continue* command causes execution to resume at a preceding *while*, i.e., the *while* that begins the smallest loop containing the *continue*.

A common loop is that of processing arguments one at a time: see *shift(1)*.

The following is a shell procedure that is also a filter. It reads a line at a time from the standard input that existed when the procedure was invoked, exiting on end-of-file.

```
while 1
    = a <-- || exit
    commands using $a ...
end
```

## SEE ALSO

*goto(1)*, *if(1)*, *onintr(1)*, *sh(1)*, *shift(1)*, *switch(1)*

## DIAGNOSTICS

while: missing end  
 while: >3 levels  
 while: syntax errors like those of *if*.  
 break: missing end  
 break: used outside loop  
 continue: used outside loop  
 end: used outside loop

## BUGS

A *goto* may be used to terminate one or more *while-end* groupings. Those who use it to branch into a loop will receive appropriately peculiar results. When an interrupt is caught and transfer to a label caused by use of *onintr(1)*, all currently effective *while-end* loops are cancelled, i.e., the *onintr* performs a *goto* that breaks all loops. Neither *while* nor *end* may be hidden behind semicolons or used within other commands.

**NAME**

**who** - who is on the system

**SYNOPSIS**

**who** [ who-file ] [ am I ]

**DESCRIPTION**

*Who*, without an argument, lists the name, terminal channel, and login time for each current UNIX user.

Without an argument, *who* examines the */etc/utmp* file to obtain its information. If a file is given, that file is examined. Typically the given file will be */usr/adm/wtmp*, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the *wtmp* file. Each login is listed with user name, terminal name (with *'/dev/'* suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with *'x'* in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, *who* behaves as if it had no arguments except for restricting the printout to the line for the current terminal. Thus *'who am I'* (and also *'who are you'*) tells you who you are logged in as.

**FILES**

*/etc/utmp*

**SEE ALSO**

*login*(1), *init*(VIII)

## NAME

write — write to another user

## SYNOPSIS

**write** user [ ttyno ]

## DESCRIPTION

*Write* copies lines from your terminal to that of another user. When first called, it sends the message

message from yourname ...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *tyno* argument may be used to indicate the last character of the appropriate terminal name.

Permission to write may be denied or granted by use of the *mesg(I)* command. At the outset writing is allowed. Certain commands, in particular *nroff* and *pr*, disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, *write* calls the Shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for that user to write back before starting to send. Each party should end each message with a distinctive signal ((o) for 'over' is conventional) that the other may reply. (oo) (for 'over and out') is suggested when conversation is about to be terminated.

## FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

## SEE ALSO

*mesg(I)*, *who(I)*, *mail(I)*

## NAME

**xargs** — construct argument list(s) and execute command

## SYNOPSIS

**xargs** [ flags ] [ command [ initial-args ] ]

## DESCRIPTION

*Xargs* combines the fixed *initial-args* with args read from standard input to execute the specified *command* one or more times. The *command* can either be executed for each line of args read, with all args read for each automatically-determined group of (at most *size* characters of) args read, or for each user-specifiable *number* of args read.

Specifically, *xargs* reads the standard input for arguments, using them to construct one or more arg lists with *initial-args* (if any), and executes *command* with each such constructed argument list; the directory containing *command*, which may also be a Shell file, must be in one's *.path* file. If *command* is omitted, */bin/echo* is used. Excepting the use of the insert option (*-i* flag, see below), arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or newlines; however, quoted strings (including embedded blanks or tabs) may also form all or part of an argument.

Excepting the *-i* option, each argument list will be constructed starting with the *initial-args*, followed by an appropriate number of arguments read from standard input. Flags *-i*, *-l*, and *-n* modify how args are selected for each command invocation; when none of these flags are coded, each arg list is built from the continuously-read args from standard input, up to *size* characters per list maximum, until there are no more args. When there are flag conflicts (e.g., *-l* vs. *-n*), the last flag has precedence. Flag values are:

- x** Causes *xargs* to terminate if any arg list would be greater than *size* characters; *-x* is forced by the options *-i* and *-l*. When neither of the options *-i*, *-l*, or *-n* are coded, the total length of all args must be within the *size* limit.
- l** *Command* is executed for each non-null line of args from standard input. A line is considered to end with the first newline *unless* the last character of the line is a blank or a tab; in either of these cases, the blank/tab signals continuation through the next non-null line. Option *-x* is forced.
- ireplstr** Insert mode: *command* is executed for each line from standard input, taking the entire line as one entity, inserting it in *initial-args* for each occurrence of *replstr*. A maximum of 5 args in *initial-args* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away, as are empty lines. Constructed args may not grow larger than 255 characters, and option *-x* is also forced. '{ }' is assumed for *replstr* if not specified.
- nnumber** Execute *command* using as many standard input args as possible, up to *number* args maximum. Fewer args will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* args remaining. If option *-x* is also coded, each *number* args must fit in the *size* limitation, else *xargs* terminates execution.
- t** Trace mode: the *command* and each constructed arg 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 instance to be executed, followed by the prompt '?...'. A reply of y (optionally followed by anything)



will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

- ssize**      The maximum total size of each arg 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 arg and the count of characters in the command name.
- eofstr**     *Eofstr* is taken as the logical end-of-file string. Underbar (`_`) is assumed for the logical EOF string if **-e** is not coded. **-e** with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical EOF string is encountered.

In args read from standard input, characters may be escaped (by a `\`) outside of quoted strings; quoted strings are stripped of the delimiting quotes, with the contents taken literally.

*Xargs* will terminate if either it receives a return code of minus one from, or if it cannot execute, *command*.

#### EXAMPLES

The following will copy 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 file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

1. `ls | xargs -p -l ar r arch`
2. `ls | xargs -p -l | xargs ar r arch`

The following will execute *com* with successive pairs of args originally typed as Shell arguments:

```
echo $* | xargs -n2 com
```

#### DIAGNOSTICS

arg list too long  
*command* not executed or returned `-1`  
Missing quote? `<string>`  
too many args with *replstr*  
insert-buffer overflow  
max arg size with insertion via *replstr* exceeded  
unknown option: `<option>`  
0 `< max-line-size <= 470: <-s option as coded>`  
#args must be positive int: `<-n option as coded>`  
can't read from tty for `-p`

**NAME**

yacc — yet another compiler-compiler

**SYNOPSIS**

yacc [ **-vrd** ] [ grammar ]

**DESCRIPTION**

*Yacc* converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output is *y.tab.c*, which must be compiled by the C compiler and loaded with any other routines required (perhaps a lexical analyzer) and the *yacc* library:

```
cc y.tab.c other.o -ly
```

If the **-v** flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

The **-r** flag causes *yacc* to accept grammars with Ratfor actions, and produce Ratfor output on *y.tab.r*. Typical usage is then

```
rc y.tab.r other.o
```

If the **-d** flag is used, the file *y.tab.h* is generated with the *define* statements that associate the *yacc*-assigned “token codes” with the user-declared “token names”. This allows source files other than *y.tab.c* to access the token codes.

**SEE ALSO**

lex(I)

*LR Parsing* by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.

*YACC — Yet Another Compiler Compiler* by S. C. Johnson.

**FILES**

y.output	
y.tab.c	
y.tab.r	when ratfor output is obtained
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/lib/liby.a	runtime library for compiler
/usr/lib/yaccpar	parser prototype for C programs
/usr/lib/yaccrpar	parser prototype for Ratfor programs

**DIAGNOSTICS**

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file.

**BUGS**

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.