UniPlus+®
System V, Release 2

Volume 1
User Manual, Section 1

UniSoft
SYSTEMS

[illegible] 1987

COPYRIGHT

DISCLAIMER

While UniSoft Systems has endeavored to exercise care in the preparation of this guide, it nevertheless makes no warranties of any kind with regard to the documentation contained herein, including no warranty of merchantability or fitness for a particular purpose. In no event shall UniSoft be liable for incidental or consequential damages in connection with or arising out of the furnishing, performance, or use of any of this documentation.

TRADEMARKS

Adapted to UniPlus$^+$ by Heather Allen of UniSoft Systems.

INTRODUCTION

This manual describes the features of System V UniPlus$^+$, a UNIX operating system. All commands, features, and facilities described in this manual are available on UniPlus$^+$.

This manual is divided into two volumes containing a total of six sections, some divided into subsections.

1. Commands and Application Programs:
   - 1. General-Purpose Commands.
   - 1C. Communications Commands.
   - 1G. Graphics Commands.
   - 1N. Networking Commands.
2. System Calls.
   - 2N. Networking Calls.
3. Subroutines:
   - 3C. C and Assembler Library Routines.
   - 3F. FORTRAN Library Routines.
   - 3M. Mathematical Library Routines.
   - 3N. Networking Routines.
   - 3S. Standard I/O Library Routines.
   - 3X. Miscellaneous Routines.
4. File Formats.
   - 4N. Networking Formats.
5. Miscellaneous Facilities.
   - 5F. Protocol Family.
   - 5P. Protocol Descriptions.
6. Games.

Section 1 (*Commands and Application Programs*) describes programs invoked directly by the user or by command language procedures, as opposed to subroutines, which are called by the user's programs. Commands generally reside in the directory **/bin** (for **bin**ary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically. Subsection 1C contains communication programs such as *cu*, *send*, *uucp*, etc.

Section 2 (*System Calls*) describes the entries into the UNIX System kernel, including the C language interface.

Section 3 (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4 (*File Formats*) documents the structure of particular kinds of files. Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories **/usr/include** and **/usr/include/sys**.

Section 5 (*Miscellaneous Facilities*) contains descriptions of character sets, macro packages, etc.

Section 6 (*Games*) describes the games and educational programs that reside in the directory **/usr/games**.

Each section consists of several entries, each a page or so long. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, except the introduction that begins each section. The page numbers of each entry start at 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

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

> NAME gives the name(s) and a brief description of the entry.
>
> SYNOPSIS summarizes the use of the program. A few conventions are used, particularly in Section 1 (*Commands*):
>
> > **Boldface** strings are typed just as they appear.
> >
> > *Italic* strings usually represent substitutable argument prototypes (such as *filename)* which you are expected to substitute for the actual name. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Square brackets [] around an argument prototype indicate that the argument is optional.

Ellipses ... show that the previous argument prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus −, plus +, or equal sign = is often taken to a 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 =.

DESCRIPTION discusses the program.

EXAMPLE(S) gives example(s) of usage.

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

SEE ALSO gives pointers to related information.

DIAGNOSTICS discusses the diagnostic indications that may be produced. Self-explanatory messages are not listed.

WARNINGS points out potential pitfalls.

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

At the front of each volume there is a a table of contents and a permuted index. The permuted index lists the commands by the information in the NAME part of each entry in the User and Administrator Manual. The permuted index contains three columns. The center column is an alphabetic list of keywords. The last column is the entry that the keyword in the center column refers to. This entry is followed by the appropriate section number in parentheses. The first column contains the remaining information from the NAME part that either precedes or follows the keyword.

For example, to look for a text editor, scan the center column for the word "editor." There are several index lines containing an "editor reference, i.e.:

ed, red: text	editor	.... ed(1)
files. ld: link	editor for common object	.... ld(1)

You can then turn to the entries listed in the last column, *ed*(1) and *ld*(1), to find information on the editor.

On most systems, all entries are available on-line via the *man*(1) command.

TABLE OF CONTENTS

1. Commands and Application Programs

COMMANDS

COMMANDS

COMMANDS

autorobots(6)	escape from the automatic robots
awk(1)	pattern scanning and processing language
back(6)	the game of backgammon
badblk(1M)	program to set or update bad block information
banner(1)	make posters
banner7(1)	print large banner on printer
basename(1)	deliver portions of path names
batch (*See at(1)*)	execute commands at a later time
bc(1)	arbitrary-precision arithmetic language
bcd(6)	convert to antique media
bcheckrc (*See brc(1M)*)	system initialization shell script
bcmp (*See bstring(3N)*)	byte string operation
bcopy (*See bstring(3N)*)	byte string operation
bcopy(1M)	interactive block copy
bdiff(1)	big diff
bessel(3M)	Bessel functions
bfs(1)	big file scanner
bind(2N)	bind a name to a socket
bj(6)	the game of black jack
blt(3C)	block transfer data
blt512 (*See blt(3C)*)	block transfer data
bool(3F)	Fortran bitwise boolean functions
boot(8)	startup procedures
brc(1M)	system initialization shell script
brk(2)	change data segment space allocation
bs(1)	a compiler/interpreter for modest-sized programs
bsearch(3C)	binary search a sorted table
bstring(3N)	bit and byte string operations
byteorder(3N)	convert values between host and network byte order
bzero (*See bstring(3N)*)	byte string operation
cabs (*See abs(3F)*)	Fortran absolute value
cal(1)	print calendar
calendar(1)	reminder service
calloc (*See malloc(3C)*)	main memory allocator
calloc (*See malloc(3X)*)	fast main memory allocator
cancel (*See lp(1)*)	cancel requests to an LP line printer
cat(1)	concatenate and print files
cb(1)	C program beautifier
cc(1)	C compiler

COMMANDS

cc5.0(1)	C compiler (System V a.out format only)
ccos (*See cos(3F)*)	Fortran cosine intrinsic function
cd(1)	change working directory
cdc(1)	change the delta commentary of an SCCS delta
ceil (*See floor(3M)*)	ceiling function
cexp (*See exp(3F)*)	Fortran exponential intrinsic function
cflow(1)	generate C flowgraph
char (*See ftype(3F)*)	explicit Fortran type conversion
chargefee (*See acctsh(1M)*)	shell procedure for accounting
chase(6)	try to escape the killer robots
chdir(2)	change working directory
checkall(1M)	faster file system checking procedure
checkcw (*See cw(1)*)	check text prepared with CW commands
checkeq (*See eqn(1)*)	check text prepared with eqn or **neqn** commands
checklist(4)	list of file systems processed by fsck
checkmm (*See mm(1)*)	check documents formatted with the MM macros
chgnod(1M)	change current UNIX system nodename
chgrp (*See chown(1)*)	change group
chmod(1)	change mode
chmod(2)	change mode of file
chown(1)	change owner
chown(2)	change owner and group of a file
chroot(1M)	change root directory for a command
chroot(2)	change root directory
ckpacct (*See acctsh(1M)*)	shell procedure for accounting
clear(1)	clear terminal screen
clearerr (*See ferror(3S)*)	stream status inquiry
clock(3C)	report CPU time used
clog (*See log(3F)*)	Fortran natural logarithm intrinsic function
close(2)	close a file descriptor
closedir (*See directory(3X)*)	flexible length directory operation
clri(1M)	clear inode
cmp(1)	compare two files
cmplx (*See ftype(3F)*)	explicit Fortran type conversion
col(1)	filter reverse line-feeds
comb(1)	combine SCCS deltas
comm(1)	select or reject lines common to two sorted files
config(1M)	configure system
conjg(3F)	Fortran complex conjugate intrinsic function

COMMANDS

connect(2N)	initiate a connection on a socket
conv(1)	object file converter
conv(3C)	translate characters
core(4)	format of core image file
cos (*See trig(3M)*)	trigonometric function
cos(3F)	Fortran cosine intrinsic function
cosh (*See sinh(3M)*)	hyperbolic function
cosh(3F)	Fortran hyperbolic cosine intrinsic function
cp(1)	copy files
cpio(1)	copy file archives in and out
cpio(4)	format of cpio archive
cpp(1)	the C language preprocessor
cpp5.0(1)	the C language preprocessor (System V a.out format only)
cpset(1M)	install object files in binary directories
craps(6)	the game of craps
crash(8)	what to do when the system crashes
creat(2)	create a new file or rewrite an existing one
cribbage(6)	the card game cribbage
cron(1M)	clock daemon
crontab(1)	user crontab file
crypt(1)	encode/decode
crypt(3C)	generate DES encryption
csh(1)	a shell (command interpreter) with C-like syntax
csin (*See sin(3F)*)	Fortran sine intrinsic function
csplit(1)	context split
csqrt (*See sqrt(3F)*)	Fortran square root intrinsic function
ct(1C)	spawn getty to a remote terminal
ctags(1)	maintain a tags file for a C program
ctermid(3S)	generate filename for terminal
ctime(3C)	convert date and time to string
ctrace(1)	C program debugger
ctype(3C)	classify characters
cu(1C)	call another UNIX system
cubic (*See ttt(6)*)	tic-tac-toe
curses(3X)	CRT screen handling and optimization package
cuserid(3S)	get character login name of the user
cut(1)	cut out selected fields of each line of a file
cw(1)	prepare constant-width text for troff
cxref(1)	generate C program cross-reference

COMMANDS

COMMANDS

eqn(1)	format mathematical text for troff
eqnchar(5)	special character definitions for eqn and neqn
erand48 (*See drand48(3C)*)	generate uniformly distributed pseudo-random numbers
erf(3M)	error function
erfc (*See erf(3M)*)	complementary error function
errdead(1M)	extract error records from dump
errdemon(1M)	error-logging daemon
errfile(4)	error-log file format
errno (*See perror(3C)*)	system error message
error(7)	error-logging interface
errpt(1M)	process a report of logged errors
errstop(1M)	terminate the error-logging daemon
etext (*See end(3C)*)	last locations in program
ex(1)	text editor
exec(2)	execute a file
execl (*See exec(2)*)	execute a file
execle (*See exec(2)*)	execute a file
execlp (*See exec(2)*)	execute a file
execv (*See exec(2)*)	execute a file
execve (*See exec(2)*)	execute a file
execvp (*See exec(2)*)	execute a file
exit(2)	terminate process
exp(3F)	Fortran exponential intrinsic function
exp(3M)	exponential function
expr(1)	evaluate arguments as an expression
exterr(1)	turn on/off the extended errors in the specified device
f77(1)	Fortran 77 compiler
fabs (*See floor(3M)*)	absolute value function
factor(1)	factor a number
false (*See true(1)*)	provide truth values
fclose(3S)	close a stream
fcntl(2)	file control
fcntl(5)	file control options
fcvt (*See ecvt(3C)*)	convert floating-point number to string
fdopen (*See fopen(3S)*)	open a stream
feof (*See ferror(3S)*)	stream status inquiry
ferror(3S)	stream status inquiry
ff(1M)	list file names and statistics for a file system
fflush (*See fclose(3S)*)	flush a stream

COMMANDS

getppid (*See getpid(2)*)	get parent process ID
getprotobyname (*See getprotoent(3N)*)	get protocol entry
getprotobynumber (*See getprotoent(3N)*)	get protocol entry
getprotoent(3N)	get protocol entry
getpw(3C)	get name from UID
getpwent(3C)	get password file entry
getpwnam (*See getpwent(3C)*)	get password file entry
getpwuid (*See getpwent(3C)*)	get password file entry
gets(3S)	get a string from a stream
getservbyname (*See getservent(3N)*)	get service entry
getservbyport (*See getservent(3N)*)	get service entry
getservent(3N)	get service entry
getsockname(2N)	get socket name
getsockopt(2N)	get options on sockets
getty(1M)	set terminal type, modes, speed, and line discipline
gettydefs(4)	speed and terminal settings used by getty
getuid(2)	get real user ID
getut(3C)	access utmp file entry
getutent (*See getut(3C)*)	access utmp file entry
getutid (*See getut(3C)*)	access utmp file entry
getutline (*See getut(3C)*)	access utmp file entry
getw (*See getc(3S)*)	get word from a stream
gmtime (*See ctime(3C)*)	convert date and time to string
graph(1G)	draw a graph
greek(1)	select terminal filter
greek(5)	graphics for the extended TTY-37 type-box
grep(1)	search a file for a pattern
group(4)	group file
grpck (*See pwck(1M)*)	group file checker
gsignal (*See ssignal(3C)*)	software signal
hangman(6)	guess the word
hashcheck (*See spell(1)*)	work with the **spell** program's hash lists
hashmake spell(1)	work with the **spell** program's hash lists
hcreate (*See hsearch(3C)*)	manage hash search tables
hdestroy (*See hsearch(3C)*)	manage hash search tables
head(1)	give first few lines
help(1)	ask for help in using SCCS
hex(1)	translates object files
hostid(1N)	set or print identifier of current host system

COMMANDS

isatty (*See ttyname(3C)*)	find name of a terminal
iscntrl (*See ctype(3C)*)	classify characters
isdigit (*See ctype(3C)*)	classify characters
isgraph (*See ctype(3C)*)	classify characters
isign (*See sign(3F)*)	Fortran transfer-of-sign intrinsic function
islower (*See ctype(3C)*)	classify characters
isprint (*See ctype(3C)*)	classify characters
ispunct (*See ctype(3C)*)	classify characters
isspace (*See ctype(3C)*)	classify characters
issue(4)	issue identification file
isupper (*See ctype(3C)*)	classify characters
isxdigit (*See ctype(3C)*)	classify characters
j0 (*See bessel(3M)*)	Bessel function
j1 (*See bessel(3M)*)	Bessel function
jn (*See bessel(3M)*)	Bessel function
join(1)	relational database operator
jrand48 (*See drand48(3C)*)	generate uniformly distributed pseudo-random numbers
kill(1)	terminate a process
kill(2)	send a signal to a process or a group of processes
killall(1M)	kill all active processes
killpg(3N)	send signal to a process group
kmem (*See mem(7)*)	core memory
l3tol(3C)	convert between 3-byte integers and long integers
l64a (*See a64l(3C)*)	convert between long integer and base-64 ASCII string
labelit (*See volcopy(1M)*)	copy file systems with label checking
last(1)	indicate last logins of users and teletypes
lastlogin (*See acctsh(1M)*)	shell procedure for accounting
lav(1)	print load average statistics
lcong48 (*See drand48(3C)*)	generate uniformly distributed pseudo-random numbers
ld(1)	link editor for common object files
ld5.0(1)	link editor (System V a.out format only)
ldaclose (*See ldclose(3X)*)	close a common object file
ldahread(3X)	read the archive header of a member of an archive file
ldaopen (*See ldopen(3X)*)	open a common object file for reading
ldclose(3X)	close a common object file
ldexp (*See frexp(3C)*)	manipulate parts of floating-point numbers
ldfcn(4)	common object file access routines
ldfhread(3X)	read the file header of a common object file
ldgetname(3X)	retrieve symbol name for object file

COMMANDS

COMMANDS

COMMANDS

pclose (*See popen(3S)*)	initiate pipe to/from a process
pdp11 (*See machid(1)*)	provide truth value about your processor type
perror(3C)	system error message
pg(1)	file perusal filter for soft-copy terminals
phys(2)	allow a process to access physical addresses
pipe(2)	create an interprocess channel
plock(2)	lock process, text, or data in memory
plot(3X)	graphics interface subroutines
plot(4)	graphics interface
pnch(4)	file format for card images
popen(3S)	initiate pipe to/from a process
pow (*See exp(3M)*)	power function
powerfail (*See brc(1M)*)	system initialization shell script
pr(1)	print files
prctmp (*See acctsh(1M)*)	shell procedure for accounting
prdaily (*See acctsh(1M)*)	shell procedure for accounting
printenv(1)	print out the environment
printf(3S)	print formatted output
prof(1)	display profile data
prof(5)	profile within a function
profil(2)	execution time profile
profile(4)	setting up an environment at login time
protocols(4N)	protocol name data base
prs(1)	print an SCCS file
prtacct (*See acctsh(1M)*)	shell procedure for accounting
ps(1)	report process status
pstat(1M)	print system facts
ptrace(2)	process trace
ptx(1)	permuted index
pty(5)	pseudo terminal driver
put(1C)	puts a file onto a remote machine
putc(3S)	put character on a stream
putchar (*See putc(3S)*)	put character on a stream
putenv(3C)	change or add value to environment
putpwent(3C)	write password file entry
puts(3S)	put a string on a stream
pututline (*See getut(3C)*)	access utmp file entry
putw (*See putc(3S)*)	put word on a stream
pwck(1M)	password file checker

pwd(1)	working directory name
qsort(3C)	quicker sort
quiz(6)	test your knowledge
rain(6)	animated raindrops display
rand(3C)	simple random-number generator
rand(3F)	Fortran uniform random-number generator
ratfor(1)	rational Fortran dialect
rc (*See brc(1M)*)	system initialization shell script
rcmd(3N)	routine for returning a stream to a remote command
rcp(1N)	remote file copy
rcvhex(1)	translates Motorola S-records from downloading into a file
read(2)	read from file
readdir (*See directory(3X)*)	flexible length directory operation
readv(3N)	read from file
real (*See ftype(3F)*)	explicit Fortran type conversion
realloc (*See malloc(3C)*)	main memory allocator
realloc (*See malloc(3X)*)	fast main memory allocator
reboot(1M)	reboot the system
reboot(2)	reboot the system
recv(2N)	receive a message from a socket
recvfrom (*See recv(2N)*)	receive a message from.a socket
recvmsg (*See recv(2N)*)	receive a message from a socket
red (*See ed(1)*)	text editor
regcmp(1)	regular expression compile
regcmp(3X)	compile a regular expression
regex (*See regcmp(3X)*)	execute a regular expression
regexp(5)	regular expression compile and match routines
reject (*See accept(1M)*)	prevent LP requests
reloc(4)	relocation information for a common object file
remque (*See insque(3N)*)	remove element from a queue
remsh(1N)	remote shell
remshd(8N)	remote shell server
reset (*See tset(1)*)	reset the teletype bits to a sensible state
rewind (*See fseek(3S)*)	reposition a file pointer in a stream
rewinddir (*See directory(3X)*)	flexible length directory operation
rexec(3N)	return stream to a remote command
rexecd(8N)	remote execution server
rlogin(1N)	remote login
rlogind(8N)	remote login server

COMMANDS

rm(1)	remove files
rmail (*See mail(1)*)	send mail to users or read mail
rmdel(1)	remove a delta from an SCCS file
rmdir (*See rm(1)*)	remove directories
robots(6)	escape from the robots
round(3F)	Fortran nearest integer functions
route(8N)	manually manipulate the routing tables
routed(8N)	network routing daemon
rresvport (*See rcmd(3N)*)	routine for returning a stream to a remote command
rsh (*See sh(1)*)	shell, the restricted command programming language
rshift (*See bool(3F)*)	Fortran bitwise boolean function
runacct(1M)	run daily accounting
ruptime(1N)	show host status of local machines
ruserok (*See rcmd(3N)*)	routine for returning a stream to a remote command
rwho(1N)	who's logged in on local machines
rwhod(8N)	system status server
sa1 (*See sar(1M)*)	system activity report package
sa2 (*See sar(1M)*)	system activity report package
sact(1)	print current SCCS file editing activity
sadc (*See sar(1M)*)	system activity report package
sag(1G)	system activity graph
sar(1)	system activity reporter
sar(1M)	system activity report package
sbrk (*See brk(2)*)	change data segment space allocation
scanf(3S)	convert formatted input
sccsdiff(1)	compare two versions of an SCCS file
sccsfile(4)	format of SCCS file
scnhdr(4)	section header for a common object file
script(1)	make typescript of terminal session
sdb(1)	symbolic debugger
sdiff(1)	side-by-side difference program
sed(1)	stream editor
seed48 (*See drand48(3C)*)	generate uniformly distributed pseudo-random numbers
seekdir (*See directory(3X)*)	flexible length directory operation
select(2N)	synchronous I/O multiplexing
semctl(2)	semaphore control operations
semget(2)	get set of semaphores
semop(2)	semaphore operation
send(2N)	send a message from a socket

sendmsg (*See send(2N)*) ....send a message from a socket
sendto (*See send(2N)*) ....send a message from a socket
services(4N) ....service name data base
setbuf(3S) ....assign buffering to a stream
setgid (*See setuid(2)*) ....set group ID
setgrent (*See getgrent(3C)*)....obtain group file entry from a group file
sethostent (*See gethostent(3N)*) ....get network host entry
sethostid (*See gethostid(2N)*)....set unique identifier of current host
sethostname (*See gethostname(2N)*) ....set name of current host
setjmp(3C) ....non-local goto
setkey (*See crypt(3C)*)....generate DES encryption
setmnt(1M) ....establish mount table
setnetent (*See getnetent(3N)*) ....get network entry
setpgrp(2) ....set process group ID
setprotoent (*See getprotoent(3N)*) ....get protocol entry
setpwent (*See getpwent(3C)*) ....get password file entry
setregid(2)....set real and effective group ID
setreuid(2) ....set real and effective user IDs
setservent (*See getservent(3N)*) ....get service entry
setsockopt (*See getsockopt(2N)*) ....set options on sockets
setuid(2)....set user ID
setutent (*See getut(3C)*) ....access utmp file entry
setvbuf (*See setbuf(3S)*) ....assign buffering to a stream
sgetl (*See sputl(3X)*) ....access long integer data in a machine independent fashion
sh(1)....shell, the standard command programming language
shl(1)....shell layer manager
shmctl(2) ....shared memory control operations
shmget(2)....get shared memory segment
shmop(2) ....shared memory operations
shutacct (*See acctsh(1M)*) ....shell procedure for accounting
shutdown(1M) ....terminate all processing
shutdown(2N)....shut down part of a full-duplex connection
sign(3F) ....Fortran transfer-of-sign intrinsic function
signal(2)....specify what to do upon receipt of a signal
signal(3F) ....specify Fortran action on receipt of a system signal
sin (*See trig(3M)*) ....trigonometric function
sin(3F)....Fortran sine intrinsic function
sinh(3F) ....Fortran hyperbolic sine intrinsic function
sinh(3M) ....hyperbolic function

COMMANDS

COMMANDS

tellnit (*See init(1M)*)	process control initialization
telldir (*See directory(3X)*)	flexible length directory operation
telnet(1N)	user interface to the TELNET protocol
telnetd(8N)	DARPA TELNET protocol server
tempnam (*See tmpnam(3S)*)	create a name for a temporary file
term(4)	format of compiled term file.
term(5)	conventional names for terminals
termcap(3X)	terminal independent operation routines
termcap(5)	terminal capability data base
terminfo(4)	terminal capability data base
termio(7)	general terminal interface
test(1)	condition evaluation command
tfind (*See tsearch(3C)*)	manage binary search trees
tftpd(8N)	DARPA Trivial File Transfer Protocol server
tgetent (*See termcap(3X)*)	terminal independent operation routine
tgetflag (*See termcap(3X)*)	terminal independent operation routine
tgetnum (*See termcap(3X)*)	terminal independent operation routine
tgetstr (*See termcap(3X)*)	terminal independent operation routine
tgoto (*See termcap(3X)*)	terminal independent operation routine
tic(1M)	terminfo compiler
time(1)	time a command
time(2)	get time
times(2)	get process and child process times
timex(1)	time a command; report process data and system activity
tmpfile(3S)	create a temporary file
tmpnam(3S)	create a name for a temporary file
toascii (*See conv(3C)*)	translate characters
tolower (*See conv(3C)*)	translate characters
touch(1)	update access and modification times of a file
toupper (*See conv(3C)*)	translate characters
tp(1)	manipulate tape archive
tplot(1G)	graphics filters
tput(1)	query terminfo database
tputs (*See termcap(3X)*)	terminal independent operation routine
tr(1)	translate characters
trek(6)	trekkie game
trig(3M)	trigonometric functions
troff(1)	typeset text
trpt(8N)	transliterate protocol trace

COMMANDS

COMMANDS

COMMANDS

write(3) .... write on a file

writev(3N) .... write on a file

wtmp (*See utmp(4)*) .... wtmp entry format

wtmpfix (*See fwtmp(1M)*) .... manipulate connect accounting records

wump(6) .... the game of hunt-the-wumpus

xargs(1) .... construct argument list(s) and execute command

xor (*See bool(3F)*) .... Fortran bitwise boolean function

xstr(1) .... extract strings from C programs to implement shared strings

y0 (*See bessel(3M)*) .... Bessel function

y1 (*See bessel(3M)*) .... Bessel function

yacc(1) .... yet another compiler-compiler

yn (*See bessel(3M)*) .... Bessel function

zabs (*See abs(3F)*) .... Fortran absolute value

PERMUTED INDEX

delivermail: deliver	mail to arbitrary people.	delivermail(8N)
netmailer: deliver	mail to B-NET.	netmailer(8N)
mail, rmail: send	mail to users or read mail.	mail(1)
processing system.	mailx: interactive message	mailx(1)
malloc, free, realloc, calloc:	main memory allocator.	malloc(3C)
/mallopt, mallinfo: fast	main memory allocator.	malloc(3X)
program. ctags:	maintain a tags file for a C	ctags(1)
regenerate groups of/ make:	maintain, update, and	make(1)
ar5.0: archive and library	maintainer.	ar5.0(1)
ar: archive and library	maintainer for portable/	ar(1)
intro: introduction to system	maintenance commands and/	intro(1M)
intro: introduction to system	maintenance procedures.	intro(8)
SCCS file. delta:	make a delta (change) to an	delta(1)
mkdir:	make a directory.	mkdir(1)
or ordinary file. mknod:	make a directory, or a special	mknod(2)
for fsck. mklost+found:	make a lost+found directory	mklost+fnd(1M)
mktemp:	make a unique filename.	mktemp(3C)
regenerate groups of/	make: maintain, update, and	make(1)
ssp:	make output single spaced.	ssp(1)
banner:	make posters.	banner(1)
session. script:	make typescript of terminal	script(1)
key.	makekey: generate encryption	makekey(1)
/realloc, calloc, mallopt,	mallinfo: fast main memory/	malloc(3X)
main memory allocator.	malloc, free, realloc, calloc:	malloc(3C)
mallopt, mallinfo: fast main/	malloc, free, realloc, calloc,	malloc(3X)
malloc, free, realloc, calloc,	mallopt, mallinfo: fast main/	malloc(3X)
entries in this manual.	man: macros for formatting	man(5)
manual.	man: print entries in this	man(1)
/tfind, tdelete, twalk:	manage binary search trees.	tsearch(3C)
hsearch, hcreate, hdestroy:	manage hash search tables.	hsearch(3C)
shl: shell layer	manager.	shl(1)
records. fwtmp, wtmpfix:	manipulate connect accounting	fwtmp(1M)
of/ ldlread, ldlinit, ldlitem:	manipulate line number entries	ldlread(3X)
frexp, ldexp, modf:	manipulate parts of/	frexp(3C)
tp:	manipulate tape archive.	tp(1)
route: manually	manipulate the routing tables.	route(8N)
/inet_netof: Internet address	manipulation routines.	inet(3N)
locate source, binary, and/or	manual for program. whereis:	whereis(1)
man: print entries in this	manual.	man(1)
for formatting entries in this	manual. man: macros	man(5)
routing tables. route:	manually manipulate the	route(8N)
ascii:	map of ASCII character set.	ascii(5)
files. diffmk:	mark differences between	diffmk(1)
umask: set file-creation mode	mask.	umask(1)
set and get file creation	mask. umask:	umask(2)
an error message file by	massaging C source. /create	mkstr(1)
table. master:	master device information	master(4)
information table.	master: master device	master(4)
regular expression compile and	match routines. regexp:	regexp(5)
math:	math functions and constants.	math(5)
constants.	math: math functions and	math(5)
eqn, neqn, checkeq: format	mathematical text for nroff or/	eqn(1)
function.	matherr: error-handling	matherr(3M)
dmax1: Fortran maximum-value/	max, max0, amax0, max1, amax1,	max(3F)
dmax1: Fortran/ max,	max0, amax0, max1, amax1,	max(3F)
max, max0, amax0,	max1, amax1, dmax1: Fortran/	max(3F)
/max1, amax1, dmax1: Fortran	maximum-value functions.	max(3F)
	maze: generate a maze.	maze(6)
maze: generate a	maze.	maze(6)
	mc68cc: C compiler.	mc68cc(1)
accounting.	mclock: return Fortran time	mclock(3F)
bcd: convert to antique	media.	bcd(6)
	mem, kmem: core memory.	mem(7)
memcpy, memset: memory/	memccpy, memchr, memcmp,	memory(3C)

NAME

intro − introduction to commands and application programs

DESCRIPTION

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

(1) Commands of general utility.
(1C) Commands for communication with other systems.
(1G) Commands used primarily for graphics and computer-aided design.

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]

where:

name	The name of an executable file.
option	−*noargletter* (*s*) or, −*argletter* < > *optarg* where < > is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with − or, − by itself indicating the standard input.

SEE ALSO

getopt(1), exit(2), wait(2), getopt(3C).

DIAGNOSTICS

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

BUGS

Regretfully, many commands do not adhere to the aforementioned syntax.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

NAME

300, 300s − handle special functions of DASI 300 and 300s terminals

SYNOPSIS

300 [+12] [−n] [−dt,l,c]

300s [+12] [−n] [−dt,l,c]

DESCRIPTION

300 supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; *300s* performs the same functions for the DASI 300s (GSI 300s or DTC 300s) 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%. *300* can be used to print equations neatly, in the sequence:

neqn file ... | nroff | 300

WARNING: if your terminal has a PLOT switch, make sure it is turned *on* before *300* is used.

The behavior of *300* 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. DASI 300 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* 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**. DASI 300 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*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like **/etc/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 **/etc/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 prevailing carriage return and line-feed delays. The *stty*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

300 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 (but not all) cases, the following sequences are equivalent:

```
nroff −T300 files ...   and   nroff files ... | 300
nroff −T300−12 files ...   and   nroff files ... | 300 +12
```

The use of *300* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *300* may produce better-aligned output.

The *neqn* names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(5).

SEE ALSO

450(1), eqn(1), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.
If your output contains Greek and/or reverse line-feeds, use a friction-feed platen 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 of text after one or more reverse line-feeds.

NAME

4014 — paginator for the Tektronix 4014 terminal

SYNOPSIS

4014 [**−t**] [**−n**] [**−cN**] [**−pL**] [file]

DESCRIPTION

The output of *4014* is intended for a Tektronix 4014 terminal; *4014* arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight-space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. TELETYPE® Teletypewriter Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page, *4014* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command !*cmd* will send the *cmd* to the shell.

The command line options are:

−t Don't wait between pages (useful for directing output into a file).

−n Start printing at the current cursor position and never erase the screen.

−c*N* Divide the screen into *N* columns and wait after the last column.

−p*L* Set page length to *L*; *L* accepts the scale factors **i** (inches) and **l** (lines); default is lines.

SEE ALSO

pr(1), tc(1), troff(1).

NAME

450 — handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

450 supports special functions of, and optimizes the use of, the DASI 450 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 *300*(1). *450* can be used to print equations neatly, in the sequence:

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

WARNING: make sure that the PLOT switch on your terminal is ON before *450* is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

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.

In many (but not all) cases, the use of *450* can be eliminated in favor of one of the following:

```
nroff −T450 files ...
```

or

```
nroff −T450−12 files ...
```

The use of *450* can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of *450* may produce better-aligned output.

The *neqn* names of, and resulting output for, the Greek and special characters supported by *450* are shown in *greek*(5).

SEE ALSO

300(1), eqn(1), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

BUGS

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen 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 of text after one or more reverse line-feeds.

NAME

acctcom − search and print process accounting file(s)

SYNOPSIS

acctcom [[options] [file]] . . .

DESCRIPTION

Acctcom reads *file*, the standard input, or **/usr/adm/pacct**, in the form described by *acct*(4) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K)**, and optionally, **F** (the *fork*/*exec* flag: **1** for *fork* without *exec*), **STAT** (the system exit status), **HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD, and BLOCKS READ** (total blocks read and written).

The command name is prepended with a **#** if it was executed with *super-user* privileges. If a process is not associated with a known terminal, a **?** is printed in the **TTYNAME** field.

If no *files* are specified, and if the standard input is associated with a terminal or **/dev/null** (as is the case when using **&** in the shell), **/usr/adm/pacct** is read; otherwise, the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file **/usr/adm/pacct** is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in **/usr/adm/pacct***?*. The *options* are:

−a Show some average statistics about the processes selected. The statistics will be printed after the output records.

−b Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.

−f Print the *fork*/*exec* flag and system exit status columns in the output.

−h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:
(total CPU time)/(elapsed time).

−i Print columns containing the I/O counts in the output.

−k Instead of memory size, show total kcore-minutes.

−m Show mean core size (the default).

−r Show CPU factor (user time)/(system-time + user-time).

−t Show separate system and user CPU times.

−v Exclude column headings from the output.

−l *line* Show only processes belonging to terminal **/dev/***line*.

−u *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a **#** which designates only those processes executed with *super-user* privileges, or **?** which designates only those processes associated with unknown user IDs.

−g *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.

−s *time* Select processes existing at or after *time*, given in the format *hr* [: *min* [: *sec*]] .

−e *time* Select processes existing at or before *time*.

−S *time* Select processes starting at or after *time*.

−E *time* Select processes ending at or before *time*. Using the same *time* for both **−S** and **−E** shows the processes that existed at *time*.

−n *pattern* Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that + means one or more occurrences.

−q Do not print any output records, just print the average statistics as with the **−a** option.

−o *ofile* Copy selected process records in the input data format to *ofile*; suppress standard output printing.

−H *factor* Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **−h** above.

−O *sec* Show only processes with CPU system time exceeding *sec* seconds.

−C *sec* Show only processes with total CPU time, system pius user, exceeding *sec* seconds.

−I *chars* Show only processes transferring more characters than the cut-off number given by *chars*.

FILES

/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO

ps(1), su(1). acct(2), acct(4), utmp(4)
acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M) in the *UniPlus$^+$ System Administrator Reference Manual.*

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time, then *time* is interpreted as occurring on the previous day.

NAME

adb – debugger

SYNOPSIS

adb [**–w**] [*objfile* [*corfile*]]

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.

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

Requests to adb are read from the standard input and responses are to the standard output. If the **–w** flag option is present, then both *objfile* and *corfile* are created (if necessary) and opened for reading and writing so that files can be modified using adb. adb ignores quit signals; an interrupt causes return to the next adb command.

To exit adb: use **$q** or **$Q** or **-d**.

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 in which it is used. If a subprocess is being debugged, then addresses are interpreted in the usual way in the address space of the subprocess. If the operating system is being debugged either post-mortem or using the special file /dev/kmem to examine interactively and/or modify memory, the maps are set to map the kernel virtual addresses. 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* A number. The prefix **0** (zero) forces interpretation in octal radix; the prefixes **0d** and **0D** force interpretation in decimal radix; the prefixes **0x** and **0X** force interpretation in hexadecimal radix. Thus **020** = **0d16** = **0x10** = 16. If no prefix appears, then the *default radix* is used; see the **$d** command. The default radix is initially hexadecimal. The hexadecimal digits are 0123456789abcdefABCDEF, with the obvious values. Note that a hexadecimal number the most significant digit of which would otherwise be an alphabetic character

must have a 0x or 0X prefix (or a leading zero, if the default radix is hexadecimal).

integer.fraction
A 32-bit floating point number.

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

<name The value of *name*, which is either a variable name or a register name. adb maintains a number of variables (see VARIABLES) named by single letters or digits. If *name* is a register name, then the value of the register is obtained from the system header in *corfile*. The register names are those printed by the $r command.

symbol A *symbol* is a sequence of upper or lowercase 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 *objfile*. An initial _ or ~ will be prefixed to *symbol*, if needed.

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

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

Monadic Operators

∗*exp* The contents of the location addressed by *exp* in *corfile*.

@*exp* The contents of the location addressed by *exp* in *objfile*.

−*exp* Integer negation.

~*exp* Bitwise complement.

#*exp* Logical negation.

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 *objfile* 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 *corfile* 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.)

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.

i *n* Disassemble the addressed instruction.
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 standard escape convention where control characters are printed as ^X and the delete character is printed as ^?.
s *n* Print the addressed characters until a zero character is reached.
S *n* Print a string using the ^X escape convention (see C above). The *n* is the length of the string including its zero terminator.
Y 4 Print 4 bytes in date format (see ctime(3)).
a 0 Print the value of dot in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.
/ global data symbol
? global text symbol
= global absolute symbol
p 4 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.

t 0 When preceded by an integer tabs to the next appropriate tab stop. For example, 8t moves to the next 8-space tab stop.

r 0 Print a space.

n 0 Print a newline.

"..." 0 Print the enclosed string.

^ Dot is decremented by the current increment. Nothing is printed.

+ Dot is incremented by 1. Nothing is printed.

– Dot is decremented by 1. Nothing is printed.

newline

Repeat the previous command with a *count* of 1.

[?/]l *value mask*

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

[?/]w *value ...*

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

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

New values for (*b1* ,*e1* ,*f1*) are recorded. If less than three expressions are given, then the remaining map parameters are left unchanged. If more than 3 expressions are given, the values of (*b2* , *e2* ,*f2*) (*b3* , *e3* ,*f3*) and so on, are changed. If the "?" or "/" is followed by "*", then the first segment (*b1* , *e1* ,*f1*) of the mapping is skipped, and the second and subsequent segments are changed instead. (There are as many (*bn* , *en* ,*fn*) triples as you have sections in your program.) If the list is terminated by "?" or "/", then the file (*objfile* or *corfile*, respectively) is used for subsequent requests. (So that, for example, "/m?" will cause "/" to refer to *objfile*.)

> *name*

Dot is assigned to the variable or register named. This command is often used in the form *constant* > *name*. This form of the command can be used to enter 96-bit "IEEE Extended Precision" numbers into the floating-point data registers fp0-fp7. For example:

```
0x3FFF00008000000000000000 > fp0
```

puts the value 1.0 into fp0. When this form of the command is used, only the first 32 bits of the constant are stored in dot. See *MC68881 Floating Point Coprocessor User's Manual* (available from Motorola Literature Distribution Center, part number MC68881UM/AD), section 2.4, "Extended Real," p. 211, for a description of IEEE Extended Precision format.

!

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

$modifier

Miscellaneous commands. The available *modifiers* are:

<*f* Read commands from the file *f*. If this command is executed in a file, further commands in the file are not seen. If *f* is omitted, the current input stream is terminated. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable *9* before the first command in *f* is executed.

<<*f* Similar to < except it can be used in a file of commands without causing the file to be closed. Variable *9* is saved during the execution of this command, and restored when it completes. There is a (small) finite limit to the number of << files that can be open at once.

>*f* Append output to the file *f*, which is created if it does not exist. If *f* is omitted, output is returned to the terminal.

? Print process ID, the signal which caused stoppage or termination, as well as the registers as $r. This is the default if *modifier* is omitted.

r Print the general registers and the instruction addressed by pc. Dot is set to pc.

f Print the floating point data registers fp0-fp7 in IEEE Extended Precision (see >*name*, above, for definition), and exponential notation, along with the floating-point control registers fpcr, fpsr, and fpiar.

b Print all breakpoints and their associated counts and commands.

c C stack backtrace. If *address* is given, then it is taken as the address of the current frame (instead of a7). 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.

d Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus 10$d never changes the default radix. To make decimal the default radix, use 0d10$d.

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 Regard all integers subsequently input as octal.

d Reset integer input as described in EXPRESSIONS.

q Exit from *adb*.

v Print all nonzero variables in octal.

m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

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

d Delete breakpoint at *address*.

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

c*s* The subprocess is continued with signal *s* **c***s* (see **signal**(3)). If *address* is given, then the subprocess is continued at this address. If no signal is specified, then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.

s*s* As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfile* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

VARIABLES

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

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

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

b The base address of the data segment.
d The data segment size.
e The entry point.
m The "magic" number (0407, 0410 , 0413).
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 *n* triples (*b1*, *e1*, *f1*), (*b2*, *e2*, *f2*), ... (*bn*, *en*, *fn*), corresponding to the number of sections in your object file, and the *file address* corresponding to a written *address* is calculated as follows.

b1≤address<e1 => *file address=address+f1−b1*, otherwise,

b2≤address<e2 => *file address=address+f2−b2*, and so on, 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 the first triple is not 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.

EXAMPLE

```
adb obj1
```

will invoke **adb** with the executable object "**obj1**" when **adb** responds with either

```
a.out file
ready
```

or

```
a.out (COFF format)
a.out a.out
ready
```

the request:

```
main,10?ia
```

will cause 16 (10 hex) instructions to be printed in assembly code, starting from location "**main**".

FILES

/bin/adb
a.out
core

SEE ALSO

a.out(4), core(4).

DIAGNOSTICS

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

BUGS

Use of # for the unary logical negation operator is peculiar.

There doesn't seem to be any way to clear all breakpoints.

In certain cases, disassembled code cannot be used directly as input to as. This is because **adb** gives more useful information than as accepts. For example, explicit register names are given in the disassembly of movm and fmovm instructions.

NAME

admin − create and administer SCCS files

SYNOPSIS

admin [**−n**] [**−i**[name]] [**−r**rel] [**−t**[name]] [**−f**flag[flag-val]] [**−d**flag[flag-val]] [**−a**login] [**−e**login] [**−m**[mrlist]] [**−y**[comment]] [**−h**] [**−z**] files

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments, which begin with −, and named files (note that SCCS file names must begin with the characters **s.**). If a named file does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

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

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

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

−i[*name*] The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **−r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty (in this case, you need to explicitly use the **−n** flag). Only one SCCS file may be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **−i** keyletter). Note that the **−i** keyletter implies the **−n** keyletter.

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

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

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

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

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

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

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

i[*str*] Causes the "No id keywords (ge6)" message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string, however the string must contain a keyword, and no embedded newlines.

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

l*list* A *list* of releases to which deltas can no longer be made (**get −e** against one of these "locked" releases fails). The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= *RELEASE NUMBER* | **a**

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

n Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

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

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

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

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

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

l*list* A *list* of releases to be "unlocked". See the **−f** keyletter for a description of the **l** flag and the syntax of a *list*.

−a*login* A *login* name, or numerical UNIX system group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters may be used on a single *admin* command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a **!** they are to be denied permission to make deltas.

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

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

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

If the comment contains spaces, you must enclose the entire comment in double quotes.

The **−y** keyletter is valid only if the **−i** and/or **−n** keyletters are specified (i.e., a new SCCS file is being created).

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

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

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

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

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

EXAMPLE

```
admin -ifile1 s.file1
```

creates a new file in SCCS format named "s.file1", from "file1".

FILES

The last component of all SCCS file names must be of the form **s.***file-name*. New SCCS files are given mode 444 (see *chmod*(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called **x.***file-name*, (see *get*(1)), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

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

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed*(1). *Care must be taken!* The edited file should *always* be processed by an **admin −h** to check for corruption followed by an **admin −z** to generate a proper check-sum. Another **admin −h** is recommended to ensure the SCCS file is valid.

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

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4).
SCCS in the *Programming Tools Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar – archive and library maintainer for portable archives

SYNOPSIS

ar key [posname] archivename [filename] ...

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 link editor. It can be used, though, for any similar purpose.

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure are described in detail in *ar*(4). The archive symbol table (described in *ar*(4)) is used by the link editor (*ld*(1)) to effect multiple passes over libraries of object files in an efficient manner. Whenever the *ar*(1) command is used to create or update the contents of an archive, the symbol table is rebuilt. The s option described below will force the symbol table to be rebuilt.

Unlike command arguments, the command key is a required part of *ar*'s command line. The key is formed with one character from the following set: **drqtpmx**. Arguments to the key are made with one or more letters from the set: **vuaibcls**. *Posname* is an archive member name used as a reference point in positioning other files in the archive. *Archivename* is the archive file. The *names* are constituent files in the archive file.

The meanings of the *key* characters are:

d Delete the named files from the archive file.

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

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

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

NAME

ar – archive and library maintainer for portable archives

SYNOPSIS

ar *key* [clsuv] [abi *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 link editor. It can be used, though, for any similar purpose.

When **ar** creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure are described in detail in ar(4). The archive symbol table described in ar(4) is used by the link editor, ld(1), to effect multiple passes over libraries of object files in an efficient manner. Whenever the ar(1) command is used to create or update the contents of an archive, the symbol table is rebuilt. The symbol table can be forced to be rebuilt by the **s** option, described below.

Key is one character from the set **dmpqrtx**. *Key* may optionally be concatenated with one or more of **clsuv** or **abi**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

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

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

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

p Print the named files in the archive.

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

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

v Verbose. Under the verbose flag 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. When used with **x**, it precedes each file with a name.

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

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

s Symbol table creation. Force the regeneration of the archive symbol table even if ar(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the strip(1) command has been used on the archive.

EXAMPLE

```
ar rc
```

creates an archive file.

FILES

/bin/ar	
/tmp/ar*	temporaries

SEE ALSO

ld(1), lorder(1), strip(1), tar(1), a.out(4), ar(4).

BUGS

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

- **p** Print the named files in the archive.
- **m** Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.
- **x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.

The meanings of the additional key arguments are as follows:

- **v** 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. When used with **x**, it precedes each file with a name.
- **c** Create. Normally *ar* creates *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.
- **l** Local. Normally *ar* places its temporary files in the directory **/tmp**. This option causes them to be placed in the local directory.
- **s** Symbol table creation. Force the regeneration of the archive symbol table even if *ar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

FILES

/tmp/ar* temporaries

SEE ALSO

ld(1), lorder(1), strip(1), a.out(4), ar(4).

BUGS

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

NAME

ar5.0 — archive and library maintainer

SYNOPSIS

ar5.0 [uvnbail] [mrxtdpq] [posname] archivename filename(s) ...

DESCRIPTION

The archive command *ar5.0* maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the loader. However, *ar5.0* can be used for any similar archiving purpose. Archives often consist of unlinked program modules.

Key is *one* character from the set **mrxtdpq**, optionally concatenated with one or more of **uvnbail**. *Archivename* is the archive file. The *filename(s)* are constituent files in or destined for the archive file. The meanings of the *key* characters are:

d Delete the named files from the archive file.

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

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

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

p Print the named files in the archive.

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

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

v Verbose. Under the verbose option, *ar5.0* 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. When used with **p**, it precedes each file with a name.

c Create. Normally *ar5.0* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.

l Local. Normally *ar5.0* places its temporary files in the directory **/tmp**. This option causes them to be placed in the local directory.

EXAMPLE

```
ar5.0 rv libar.a text.o
```

places file "text.o" in archive "libar.a".

```
ar5.0 bm file1 archivename file2
```

changes the location of a file inside an archive. "File2" is the file to be moved. "File2" is moved to a new position before "file1".

FILES

/tmp temporaries

SEE ALSO

ld5.0(1), ar5.0(4).

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.
Sufficient disk space must be present to make an entire copy of the archive or the *ar5.0* command will fail.

NAME

as — common assembler

SYNOPSIS

as [**−o** objfile] [**−n**] [**−m**] [**−R**] [**−V**] filename

DESCRIPTION

The *as* command assembles the named file. The following flags may be specified in any order.

−o *objfile* Put the output of assembly in *objfile*. By default, the output filename is formed by removing the **.s** suffix, if there is one, from the input filename and appending a **.o** suffix.

−n Turn off long/short address optimization. By default, address optimization takes place.

−m Run the *m4* macro pre-processor on the input to the assembler.

−R Remove (unlink) the input file after assembly is completed. This option is off by default.

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

FILES

/usr/tmp/as[1-6]*XXXXXX* temporary files

SEE ALSO

ld(1), m4(1), nm(1), strip(1), a.out(4).

WARNING

If the **−m** (*m4* macro pre-processor invocation) option is used, keywords for *m4* (see *m4*(1)) cannot be used as symbols (variables, functions, labels) in the input file because *m4* cannot determine which are assembler symbols and which are real *m4* macros.

BUGS

Arithmetic expressions are permitted to have only one forward referenced symbol per expression.

NAME

as – common assembler

SYNOPSIS

as [**–m**] [**–n**] [**–o** *objfile*] [**–R**] [**–V**] *filename*

DESCRIPTION

The **as** command assembles *filename*. The following flags may be specified in any order.

–o *objfile* Put the output of assembly in *objfile*. By default, the output filename is formed by removing the **.s** suffix, if there is one, from the input filename and appending a **.o** suffix.

–n Turn off long/short address optimization. By default, address optimization takes place.

–m Run the **m4** macro pre-processor on the input to the assembler.

–R Remove (unlink) the input file after assembly is completed. This flag option is off by default.

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

FILES

/bin/as
/usr/tmp/as[1-6]*XXXXXX* temporary files

SEE ALSO

adb(1), ld(1), m4(1), nm(1), strip(1), a.out(4).

WARNING

If the **–m** (**m4** macro preprocessor invocation) flag option is used, keywords for **m4** cannot be used as symbols (variables, functions, labels) in the input file because **m4** cannot determine which are assembler symbols and which are real **m4** macros. See**m4**(1).

BUGS

Arithmetic expressions are permitted to have only one forward referenced symbol per expression.

NAME

as5.0 − assembler

SYNOPSIS

as5.0 [−oobjfile] [−**v**] [−**l**] [name ...]

DESCRIPTION

As5.0 assembles the named files, or the standard input if no file name is specified.

All undefined symbols in the assembly are treated as global.

The relocatable output of the assembly is left on the file *objfile*; if that is omitted, *a.out5.0* is used.

The −**v** option interprets 68010 instruction mnemonics.

The −**l** option produces an assembly listing on file *objfile.lst*. If the −**l** option is specified and no −**o** parameter is specified, the assembly listing is placed on *a.lst*.

EXAMPLE

```
as5.0 −o file.o filea fileb filec
```

would assemble the three named files and put the output of the assembly into "file.o".

FILES

/tmp/as*	default temporary file
a.out	default resultant object file
a.lst	default assembly listing file

SEE ALSO

ld5.0(1), nm5.0(1), a.out5.0(4)
AS in the *Programming Guide*.

NAME

asa — interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

Asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names are supplied. The first character of each line is assumed to be a control character; their meanings are:

' ' (blank) single new line before printing

0 double new line before printing

1 new page before printing

+ overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ' '. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic will appear on standard error. This program forces the first line of each input file to start on a new page.

EXAMPLE

To correctly view the output of FORTRAN programs which use ASA carriage control characters, *asa* could be used as a filter, thus:

```
a.out | asa | lp
```

and the output, properly formatted and paginated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by:

```
asa file
```

SEE ALSO

efl(1), f77(1), fsplit(1) ratfor(1).

NAME

at, batch – execute commands at a later time

SYNOPSIS

at time [date] [+increment]
at –r job ...
at –l [job ...]

batch

DESCRIPTION

At and *batch* read commands from standard input to be executed at a later time. *At* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. *At* **–r** removes jobs previously scheduled with *at*. The **–l** option reports all jobs scheduled for the invoking user.

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

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

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

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

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

Thus legitimate commands include:

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

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

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

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

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *Sh*(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

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

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

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

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

```
echo "sh shellfile" | at 1900 thursday next week
```

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

kill(1), mail(1), nice(1), ps(1), sh(1), cron(1M).

DIAGNOSTICS

Complaints about various syntax errors and times out of range.

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [−Fc] [prog] [parameters] [files]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as **−f** *file*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

Parameters, in the form x=... y=... etc., may be passed to *awk*.

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

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

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

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

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, −, * , /, %, and concatenation (indicated by a blank). The C operators ++, −−, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

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

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr* (*s* ,*m* ,*n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf* (*fmt* ,*expr* ,*expr* ,...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, ||, **&&**, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see *grep*(1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

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

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

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

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

or by using the **−F***c* option.

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

EXAMPLE

```
awk "length > 72" filea
```

prints lines longer than 72 characters on the standard output.

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

prints the first two fields of each line in opposite order.

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

adds up the first column and prints the sum and average.

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

prints all the fields of each line in reverse order. The output prints one field per line.

```
awk "/start/, /stop/" filea
```

prints all lines between start/stop pattern pairs, for every such pair in the file.

SEE ALSO

grep(1), lex(1), sed(1) malloc(3X).
AWK in the *Programming Tools Guide*.

BUGS

Input white space is not preserved on output if fields are involved.
There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

NAME

banner — make posters

SYNOPSIS

banner strings

DESCRIPTION

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

EXAMPLE

```
banner asa
```

will cause the characters "a", "s" and "a" to be printed as large letters on the screen.

SEE ALSO

echo(1).

NAME

banner7 — print large banner on printer

SYNOPSIS

banner7 [**−w***n*] message ...

DESCRIPTION

Banner7 prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If **−w** is given, the output is scrunched down from a width of 132 to *n*, suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS

Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The **−w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

AUTHOR

Mark Horton

NAME

basename, dirname − deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

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

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

EXAMPLE

Invoked with the argument **/usr/src/cmd/cat.c,**

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

compiles the named file and moves the output to a file named "cat" in the current directory.

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

sets the shell variable NAME to **/usr/src/cmd.**

SEE ALSO

sh(1).

BUGS

The *basename* of / is null and is considered an error.

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

bc [−c] [−l] [file ...]

DESCRIPTION

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

```
Comments
      are enclosed in /* and */.

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

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

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

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

Function definitions
      define L ( L ,..., L ) {
              auto L, ... , L
              S; ... S
              return ( E )
      }

Functions in −l math library
      s(x)    sine
      c(x)    cosine
      e(x)    exponential
```

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

All function arguments are passed by value.

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

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

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

EXAMPLE

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

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

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

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

FILES

/usr/lib/lib.b	mathematical library
/usr/bin/dc	desk calculator proper

SEE ALSO

dc(1).
BC in the *Programming Tools Guide*.

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 file1 file2 [n] [−s]

DESCRIPTION

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

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

EXAMPLE

```
diff file1 file2
```

where "file1" and "file2" are two versions of the manual text for the *cp* command, produces:

```
22c22
< .IR sh (1)
---
> .IR sh\^ (1)
35c35
< .IR chmod (2))
---
> .IR chmod\^ (2))
50a51,56
> .SH EXAMPLE
> .IP
> cp alpha beta gamma /unisoft/roxanne
> .PP
> places copies of the three files in the directory
> .BR /unisoft/roxanne .
```

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*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional − suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed as in *ed*. 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. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

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

xf *file*

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

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

xo [*file*]

Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label*

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

(. , .)**xb/** *regular expression/ label*

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

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

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

```
xb/^/ label
```

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

xt *number*

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

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

The variable name is the specified *digit* following the **xv**. **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. **Xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a % in front of the variable name. For example, using the above assignments for variables **5** and **6**:

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

will all print the first 100 lines.

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

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

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

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.
Another feature of the **xv** command is that the first line of output from a UNIX system command can be stored into a variable. The only requirement is that the first character of *value* be an !. For example:

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

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

```
xv7\!date
```

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

xbz *label*

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

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

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

EXAMPLE

```
bfs text
```

will invoke *bfs* with the file named "text".

SEE ALSO

csplit(1), ed(1), regcmp(3X).

DIAGNOSTICS

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

NAME

bs – a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [*file* [*args*]]

DESCRIPTION

bs is a remote descendant of Basic and Snobol4 with a little C language thrown in. **bs** is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the **trace** and **dump** statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see **compile** and **execute** below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

bs programs are made up of input lines. If the last character on a line is a \, the line is continued. **bs** accepts lines of the following form:

statement
label statement

A *label* is a *name* (see below) followed by a colon. A label and a variable may have the same name.

A **bs** statement is either an expression or a keyword followed by zero or more expressions. Some keywords (**clear**, **compile**, **!**, **execute**, **include**, **ibase**, **obase**, and **run**) are always executed as they are compiled.

Statement Syntax

expression

The *expression* is executed for its side effects (value, assignment or function call). The details of expressions follow the description of statement types below.

break

break exits from the inner-most **for/while** loop.

clear

Clears the symbol table and compiled statements. **clear** is executed immediately.

compile [*expression*]

Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a filename for further input. A **clear** is associated with this latter case. **compile** is executed immediately.

continue

continue transfers to the loop-continuation of the current **for/while** loop.

dump [*name*]

The name and current value of every non-local variable is printed. Optionally, only the named variable is reported. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.

exit [*expression*]

Return to system level. The *expression* is returned as process status.

execute

Change to immediate execution mode (an interrupt has a similar effect). This statement does not cause stored statements to execute (see **run** below).

for *name* = *expression expression statement*
for *name* = *expression expression*

...

next

for *expression* , *expression* , *expression statement*
for *expression* , *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. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

fun f([*a*, ...]) [*v*,
nuf

fun defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn

A way to signal the failure of a user-written function. See the interrogation operator (?), below. If interrogation is not present, **freturn** merely returns zero. When interrogation *is* active, **freturn** transfers to that expression (possibly by-passing intermediate function returns).

goto name

Control is passed to the internally-stored statement with the matching label.

ibase *N*

ibase sets the input base (**radix**) to *N*. The only supported values for *N* are **8**, **10** (the default), and **16**. Hexadecimal values 10–15 are entered as a–f. A leading digit is required (i.e., **f0a** must be entered as **0f0a**). **ibase** (and **obase** below) are executed immediately.

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. The strings **0** and "" (null) evaluate as zero. In the second form, an optional **else** allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an **else** is an **if**; only other **fi**'s can be on the same line with a **fi**. The elision of **else** and **if** into an **elif** is supported. Only a single **fi** is required to close an **if**...**elif**...[**else**...] sequence.

include *expression*

The *expression* must evaluate to a filename. The file must contain **bs** source statements. Such statements become part of the program being compiled. **include** statements may not be nested.

obase *N*

obase sets the output base to *N* (see **ibase** above).

onintr *label*
onintr

The **onintr** command provides program control of interrupts. In the first form, control will pass to the label given, just as if a **goto** had been executed at the time **onintr** was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause **bs** to terminate.

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 random number generator is reset. Control is passed to the first internal statement. If the **run** statement is contained in a file, it should be the last statement.

stop

Execution of internal statements is stopped. **bs** reverts to immediate mode.

trace [*expression*]

The **trace** statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each **return** decrements the **trace** expression value.

while *expression statement*
while *expression*
. . .
next

while is similar to **for** except that only the conditional expression for loop-continuation is given.

! *shell command*

An immediate escape to the Shell.

. . .

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

Expression Syntax

name

A *name* is used to specify a variable. Names are composed of a letter (upper or lower case), optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in **fun** statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function **open()** below).

name ([*expression* [, *expression*]. . .])

Functions may be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a **fun** statement. Arguments to functions are passed by value.

name [*expression* [,*expression*]. . .]

This syntax is used to reference either arrays or tables (see built-in **table** functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference 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.

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.

string

Character strings are delimited by " *characters*. The \ escape character allows the double quote (\"), newline (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. Otherwise, \ stands for itself.

(*expression*)

Parentheses are used to alter the normal order of evaluation.

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

The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

(False, True)[a == b]

has the value **true** if the comparison is true.

? *expression*

The interrogation operator tests for the success of the expression, rather than its value. At the moment, it is useful for testing for end-of-file (see examples in PROGRAMMING TIPS, below), the result of the **eval** built-in function, and for checking the return from user-written functions (see **freturn**).

An interrogation *trap* (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

− *expression*
The result is the negation of the expression.

++ *name*
Increments the value of the variable (or array reference). The result is the new value.

-- *name*
Decrements the value of the variable. The result is the new value.

! *expression*
The logical negation of the expression. Watch out for the shell escape command.

expression operator expression
Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators
(in increasing precedence):

=
= 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.

_
_ (underscore) is the concatenation operator.

& |
& (logical and) has result zero if either of its arguments are zero. It has result one if both of 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 is non-zero. Both operators treat a null string as a 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*. A string comparison is made if both operands are strings.

+ −

Add and subtract.

*** / %**

Multiply, divide, and remainder.

^

Exponentiation.

BUILT-IN FUNCTIONS

Dealing with arguments

arg(*i*)

is the value of the *i*-th actual parameter on the current level of function call. At level zero, arg returns the *i*-th command-line argument (**arg**(0) returns **bs**).

narg( )

returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

abs(*x*)

is the absolute value of *x*.

atan(*x*)

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

ceil(*x*)

returns the smallest integer not less than *x*.

cos(*x*)

is the cosine of *x* (radians).

exp(*x*)

is the exponential function of *x*.

floor(*x*)

returns the largest integer not greater than *x*.

log(*x*)

is the natural logarithm of *x*.

rand( )

is a uniformly distributed random number between zero and one.

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

sqrt(*x*)
is the square root of *x*.

String operations

size(*s*)
the size (length in bytes) of *s* is returned.

format(*F*, *a*)
returns the formatted value of *a*. *F* is assumed to be a format specification in the style of **printf**(3S). Only the %. . .f, %. ..e, and %. . .s types are safe.

index(*x*, *y*)
returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

trans(*s*,*f*, *t*)
Translates characters of the source *s* from matching characters in *f* to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

substr(*s*, *start*,*width*)
returns the sub-string of *s* defined by the *start*ing position and *width*.

match(*string*, *pattern*)
mstring(*n*)
The *pattern* is similar to the regular expression syntax of the **ed**(1) command. The characters ., [,], ^ (inside brackets), * and $ are special. The **mstring** function returns the *n*-th (1 <= *n* <= 10) substring of the subject that occurred between pairs of the pattern symbols \(and \) for the most recent call to **match**. To succeed, patterns must match the beginning of the string (as if all patterns began with ^). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a–z]\)") == 6
mstring(1) == "b"
```

File Handling

open(*name*, *file*, *function*
close(*name*)
The *name* argument must be a **bs** variable name (passed as a string). For

the **open**, the *file* argument may be i.) a **0** (zero), **1**, or **2**, representing standard input, output, or error output, respectively, ii.) a string representing a filename, or iii.) a string beginning with an **!** representing a command to be executed (via **sh −c**). The *function* argument must be either r (read), **w** (write), **W** (write without newline), or **a** (append). After a **close**, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

access(*s*, *m*)
executes **access**(2).

ftype(*s*)
returns a single character file type indication: **f** for regular file, **p** for FIFO (i.e., named pipe), **d** for directory, **b** for block special, or **c** for character special.

Tables

table(*name*, *size*)
A *table* in **bs** is an associatively accessed, single-dimension array. Subscripts (called *keys*) are strings (numbers are converted). The *name* argument must be a **bs** variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. **bs** prints an error message and stops on table overflow.

item(*name*, *i*)

key()
The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the subscript of the previous *item* call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table, for example:

```
table("t", 100)
...
# If word contains the string "party",
# the following expression adds one
# to the count of that word:
```

```
++t[word]
 . . .
# To print out the the key/value pairs:
for i = 0, ?(s ++iif key() put= key()_ : _s
```

iskey(*name*, *word*)

The **iskey** function tests whether the key *word* exists in the table *name* and returns one for true, zero for false.

Odds and ends

eval(*s*)

The string argument is evaluated as a **bs** expression. The function is handy for converting numeric strings to numeric internal form. **eval** may also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++"_ name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control **bs** error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named XXX (instead of halting the user's program). The following executes a **goto** to the label *L* (if it exists):

```
label="L"
if !(?eval("goto "_ label)) puterr = "no label"
```

plot(*request*, *args*)

The **plot** function produces output on devices recognized by **tplot**(1G). The *requests* are as follows:

Call	Function
plot(0, *term*)	causes further **plot** output to be piped into **tplot**(1G) with an argument of −**T***term*.
plot(1)	erases the plotter.
plot (2, *string*)	labels the current point with *string*.
plot(3, *x1*,*y1*,*x2*,*y2*)	draws the line between (*x1*,*y1*) and (*x2*,*y2*).
plot(4, *x*,*y*,*r*)	draws a circle with center (*x*,*y*) and radius *r*.

plot(5, *x1* **,** *y1* **,** *x2* **,** *y2* **,** *x3* **,** *y3* **)**
draws an arc (counterclockwise) with center (*x1* ,*y1*) and endpoints (*x2* ,*y2*) and (*x3* ,*y3*).

plot(6) is not implemented.

plot(7,*x***,***y***)** makes the current point (*x*,*y*).

plot(8,*x***,***y***)** draws a line from the current point to (*x*,*y*).

plot(9,*x***,***y***)** draws a point at (*x*,*y*).

plot(10,*string***)** sets the line mode to *string*.

plot(11,*x1***,***y1***,***x2***,***y2***)** makes (*x1* ,*y1*) the lower left corner of the plotting area and (*x2* ,*y2*) the upper right corner of the plotting area.

plot(12,*x1***,***y1***,***x2***,***y2***)** causes subsequent *x*(*y*) coordinates to be multiplied by *x1* (*y1*) and then added to *x2* (*y2*) before they are plotted. The initial scaling is **plot(**12,1.0,1.0,0.0,0.0**)**.

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to **tplot**(1G). See **plot**(4) for more details.

last( )

in immediate mode, **last** returns the most recently computed value.

PROGRAMMING TIPS

Using **bs** as a calculator:

```
$ bs
#   Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

#   Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4  bal = bal + bal*int
bal − 1000
346.855007
...
```

```
exit
```

The outline of a typical bs program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while ?(str = read)
        ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
#   Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

#   Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

EXAMPLE

```
bs program 1 2 3
```

compiles and/or executes the file named **program** as well as statements typed from standard input. The arguments 1, 2, and 3 are passed as arguments to the compiled/executed program.

FILES

/bin/bs

SEE ALSO

ed(1), ksh(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), plot(4).
See *UniPlus+ Programming Tools* for further description of the mathematical functions (**pow** on **exp**(3M) is used for exponentiation); bs uses the Standard Input/Output package.

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. If neither is specified, a calendar for the present 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.

EXAMPLE

```
cal 9 1752
```

produces a calendar for September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.
Beware that "cal 83" refers to the early Christian era, not the 20th century.

NAME

calendar — reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

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

When an argument is present, *calendar* does its job for every user who has a file **calendar** in their login directory and sends them any positive results by *mail*(1). Normally this is done daily by facilities in the UNIX operating system under control of *cron*(1M).

EXAMPLE

If the user has the following line, among other lines containing date information, in the file "calendar" in the login directory:

```
        Monday, September 6   Labor Day Holiday
```

typing in

```
        calendar
```

either on the Friday before or on the specified Monday will cause this line to be printed on the screen.

FILES

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

SEE ALSO

mail(1).

BUGS

Your calendar must be public information for you to get reminder service. *Calendar's* extended idea of "tomorrow" does not account for holidays.

SEE *LP*

NAME

cat − concatenate and print files

SYNOPSIS

cat [**−u**] [**−s**] [**−v** [**−t**] [**−e**]] file ...

DESCRIPTION

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

If no input file is given, or if the argument − is encountered, *cat* reads from the standard input file. Output is buffered unless the **−u** option is specified. The **−s** option makes *cat* silent about non-existent files.

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

When used with the **−v** option, **−t** causes tabs to be printed as ^I's, and **−e** causes a **$** character to be printed at the end of each line (prior to the new-line). The **−t** and **−e** options are ignored if the **−v** option is not specified.

EXAMPLE

```
cat file
```

prints the file, and:

```
cat file1 file2 > file3
```

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

WARNING

Command formats such as

```
cat file1 file2 >file1
```

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

SEE ALSO

cp(1), pg(1), pr(1).

NAME

cb − C program beautifier

SYNOPSIS

cb [**−s**] [**−j**] [**−l** leng] [file ...]

DESCRIPTION

Cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. Under default options, *cb* preserves all user new-lines. Under the **−s** flag *cb* canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The **−j** flag causes split lines to be put back together. The **−l** flag causes *cb* to split lines that are longer than *leng*.

EXAMPLE

If there is a C program called *test.c* which looks like this:

```
#define COMING 1
#define GOING 0

main ()
{
/*  This is a test of the C Beautifier  */
if (COMING)
printf ("Hello, world\n");
else
printf ("Goodbye, world\n");
}
```

Then using the *cb* command as shown below produces the output shown:

```
cb test.c
#define COMING 1
#define GOING 0

main ()
{
        /*  This is a test of the C Beautifier  */
        if (COMING)
                printf ("Hello, world\n");
        else
                printf ("Goodbye, world\n");
}
```

SEE ALSO

cc(1).
The C Programming Language by B. W. Kernighan and D. M. Ritchie.

BUGS

Punctuation that is hidden in preprocessor statements will cause indentation errors.

NAME

cc – C compiler

SYNOPSIS

cc [**–B** *string*] [**–c**] [**–C**] [**–D** *symbol*[=def]] [**–E**] [**–fm68881**] [**–g**] [**–I** *dir*] [**–L** *dir*] [**–l** x] [**–n**] [**–o** *outfile*] [**–O**] [**–p**] [**–P**] [**–R**] [**–s**] [**–S**] [**–t** [p012al]] [**–T**] [**–U** *symbol*] [**–v**] [**–W** *c,arg1* [*,arg2*...] [**–Z** *flags*] [**–#**] ... *files* ...

DESCRIPTION

The cc command is a front-end program that invokes the preprocessor, compiler, assembler and linkage editor, as appropriate. (The default is to do them all). It generates assembly instructions. cc accepts the following 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 in a 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, no *.o* is produced. In the same way, arguments whose names end with *.s* are taken to be assembly source programs and are assembled to produce a *.o* file.

The following flag options are interpreted by cc. (Other flag options may be passed to the assembler and the linker. See ld(1) for link editor flag options and as(1) for assembler options.)

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

–C Pass along all comments except those found on cpp(1) directive lines. The default strips out *all* comments.

–p Arrange for the compiler to produce code which counts the number of times each routine is called. Also, if link editing takes place, replace the standard startoff routine by one which automatically calls monitor(3C) at the start and arranges to write out a mon.out file at normal termination of execution of the object program.

–fm68881
Generate inline code for MC68881 floating point coprocessor. When using this option, the library **–lc881** must be included on the command line. If functions from the math library are needed, then **–lm881** must be included as well.

–g Generate additional information needed for the use of sdb(1).

–l *x* Same as –l in ld(1). Search a library, lib*x*.a, where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a –l is significant. By default, libraries are located in LIBDIR. If you plan to use the –L (see below) option, that option *must* precede –l on the command line.

–L *dir* Same as **–L** in **ld**(1). Change the algorithm of searching for **libx.a** to look in *dir* before looking in LIBDIR. This option is effective only if it precedes the **–l** option on the command line.

–o *outfile*

Same as **–o** in **ld**(1). Produce an output object file, *outfile*. The default name of the object file is **a.out**.

–O Invoke an object-code optimizer. The optimizer moves, merges, and deletes code, so symbolic debugging with line numbers could be confusing when the optimizer is used. This option may not work properly on code containing **asm** directives.

–R Have assembler remove its input file when done.

–W*c,arg1*[*,arg2...*]

Hand off the argument(s) *argi* (where *i* = 1,2, . . . *n*) to pass *c*, where *c* is one of [p012al] indicating preprocessor, compiler first pass, compiler second pass, optimizer, assembler, or link editor, respectively. For example:

–W*a,-m*

Invoke the **m4** macro preprocessor on the input to the assembler. (The **–m** flag option to **as** causes it to go through **m4**.) This must be done for a source file that contains assembler escapes.

–s

Same as **–s** in **ld**(1). Strip line number entries and symbol table information from the output of object file.

–S

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

–t [p012al]

Find only the designated preprocessor (**p**), compiler (**0** and **1**), optimizer (**2**), assembler (**a**) and link editor (**l**) passes whose names are constructed with the *string* argument to the **–B** flag option. In the absence of a **–B** option and its argument, *string* is taken to be **/lib/n.** Using **–t** with no argument is equivalent to **–tp012**.

–T

Truncate symbol names to 8 significant characters. Many modern C compilers, as well as the proposed ANSI standard for C, allow arbitrary length variable names. **cc** follows this convention. The **–T** option is provided for compatibility with earlier systems.

–E

Run only **cpp**(1) on the named C programs, and send the result to the standard output.

–P

Run only **cpp**(1) on the named C programs, and leave the result on corresponding files suffixed .i.

−D*symbol*[=*def*]
Define the external *symbol* to the preprocessor and give it the value *def* (if specified). If not *def* is given, *symbol is defined as 1*. This mechanism is useful with the conditional statements in the preprocessor by allowing symbols to be defined external to the source file.

−U*symbol*
Undefine *symbol* to the preprocessor.

−I*dir*
Search for #include files (whose names do not begin with /) in *dir* before looking in the directories on the standard list. Thus, *#include* files whose names are enclosed in double quotes are searched for first in the directory of the .c file currently being compiled, then in directories named in −I flag options, and last in directories on a standard list. For *#include* files whose names are enclosed in <>, the directory of the *file* argument is not searched.

−B*string*
Construct pathnames for substitute preprocessor, compiler, assembler, and link editor passes by concatenating *string* with the suffixes cpp, comp, optim, as, and ld. If *string* is empty it is taken to be /lib/o.

−v
Print the command line for each subprocess executed.

−n
Arrange for the loader to produce an executable which is linked in such a manner that the text can be made read-only and shared (nonvirtual) or paged (virtual).

−#
Special debug option which, without actually starting the program, echoes the names and arguments of subprocesses which would have started.

−Z*flags*
Special *flags* to override the default behavior (see NOTES). Currently recognized *flags* are:

c	suppress returning pointers in both a0 and d0
n	emit no code for stack growth
m	use Motorola SGS compatible stack growth code
p	use tst.b stack probes
E	ignore all environment variables
F	flip byte order in multicharacter constants.
I	emit inline code for MC68881 floating point coprocessor
l	suppress selection of a loader command file
t	do not delete temporary files
S	compile to be SVID-compatible.

Other arguments are taken to be either link editor flag option arguments or C-compatible object programs, typically produced by an earlier cc run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are link-edited (in the order given) to produce an executable program with the name **a.out** unless the **–o** flag option of the link editor is used.

FILES

/usr/bin/cc	
file.c	input file
file.o	object file
file.s	assembly language file
a.out	link-edited output
/usr/tmp/mc68?	temporary
/lib/cpp	preprocessor
/lib/c0	compiler, first pass **/lib/c1** compiler, second pass
/lib/optim	optimizer
/bin/as	assembler, as(1)
/bin/ld	link editor, ld(1)
/lib/libc.a	standard library, see (3)
/usr/lib/shared.ld	loader command file for shared text or paged programs
/usr/lib/unshared.ld	loader command file for unshared text programs

SEE ALSO

as(1), dis(1), ld(1).
The C Programming Language by B. W. Kernighan and D. M. Ritchie, Prentice–Hall, 1978.

DIAGNOSTICS

The diagnostics produced by the C compiler are sometimes cryptic. Occasional messages may be produced by the assembler or link editor.

WARNING

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value are to call exit(2) explicitly or to leave the function main() with a *return expression;* construct.

NOTES

This version of cc is based on the cc released with the Motorola SGS. It has been changed in the following ways:

- The **–Z** flag option has been added to explicitly control generation of stack growth code for cross development environments or generation of standalone code. The Motorola SGS looks for an environment variable called M68000 and generates stack growth code if the variable is set to STACKCHECK. This cc defaults to stack probes on 68000 host processors and no stack growth code on the 68020 processors.

- The default is to produce shared text programs. To produce nonshared text programs, you must run ld.
- When used with the −g flag option, the arguments −u _dbargs −lg will be inserted in the command line for the link phase. This causes the contents of libg.a to be linked in. Note that the Motorola SGS only generates the loader argument −lg, which is not sufficient to cause loading of the library's contents.
- The −v (verbose) flag option has been added to print the command line for each subprocess executed. This helps to isolate problems to a specific phase of the compilation process by showing exactly what cc is doing, so that each phase can be run by hand if necessary.
- The Motorola SGS compiler expects functions returning pointers or structures to return their values in a0, and other functions to return their values in d0/d1. Because of the large body of existing code that has inconsistent type declarations, the version of the compiler released with the generic 68000 port emits code to return pointers in both a0 and d0 by copying a0 to d0 just prior to returning. This copy operation can be suppressed with the −Zc flag option, thus generating slightly smaller code.

NAME

cc5.0 − C compiler

SYNOPSIS

cc5.0 [option] ... file ...

DESCRIPTION

Cc5.0 is the UNIX C compiler.

Cc5.0 accepts several types of arguments:

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

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

The following options are interpreted by *cc5.0*. See *ld5.0*(1) for link editor options.

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

−n Passed on to *ld5.0* to make the text of the resulting program shared.

−p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if link editing takes place, replace the standard startup routine by one which automatically calls *monitor*(3C) 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).

−fsky Use the sky floating point library.

−O(BKPS)
Invoke an object-code improver (optimizer). If **B** is specified, "jump to subroutine" instructions are changed to "branch to subroutine" instructions (where possible). If **K** is specified, certain UNIX kernel optimizer functions are not performed. If P is specified, stack probe instructions are removed. (Note: **P** should only be used for the operating system source.) If S is specified, stack frame optimization is performed.

−R (addr)
Pass on to *ld5.0*, making the resulting object module *origin*ed at *addr(hex)*.

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

−E Run only *cpp5.0*(1) on the named C programs, and send the result to standard output.

−P Run only the macro preprocessor on the named C programs, and send the result to the corresponding files suffixed. '.i'

−C Prevent the macro preprocessor from eliding (leaving out) comments.

−o *output* Name the final executable output file *output*. If this option is used the file "a.out" will be left undisturbed.

−D*name*=*def*
−D*name* Define the *name* to the preprocessor, as if by **#define**. If no definition is given, the name is defined as "1".

−U*name* Remove any initial definition of *name*.

−I*dir* **#include** files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in −I options, then in the directory **/usr/include**.

−v print the name of each subprocess as it is executing.

Other arguments are taken to be either link editor option arguments, or C-compatible object programs, typically produced by an earlier *cc5.0* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked via *ld5.0*(1) (in the order given) to produce an executable program with name *a.out*.

EXAMPLE

```
cc5.0 −o output prog1.c prog2.c prog3.c
```

would compile code in the three named C programs and put the compiled code into the file "output".

FILES

file.c	input file
file.o	object file
a.out	linked output
/tmp/ctm?	temporary
/lib/5.0/cpp	preprocessor
/lib/5.0/c	combined compiler pass1 and pass2
/lib/5.0/c2	optional optimizer invoked with "-O"
/lib/5.0/crt0.o	runtime startoff
/lib/5.0/mcrt0.o	runtime startoff for profiling
/lib/5.0/libc.a	standard library, see section 3
/usr/include/5.0	standard directory for '#include' files
/lib/5.0/libm.a	math library
/lib/5.0/libsky.a	sky floating point routines
/lib/5.0/crt0sky.o	runtime startoff using sky
/lib/5.0/mcrt0sky.o	runtime startoff for profiling using sky

SEE ALSO

ld5.0(1), lint(1), prof(1), monitor(3C)
The C Programming Language, Prentice-Hall, 1978, by B. W. Kernighan and D. M. Ritchie; *Programming Guide*.

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor. Confusing syntax may cause the C compiler to indicate an error on the line following the actual error.

NAME

cd — change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **$HOME** is used as the new working directory. If *directory* specifies a complete path starting with /, ., .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable. *Cd* must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and internal to the shell.

EXAMPLE

```
cd /unisoft/usr/games
```

would relocate you to the directory **/unisoft/usr/games** if this directory is executable (searchable) by you.

SEE ALSO

pwd(1), sh(1), chdir(2).

NAME

cdc − change the delta commentary of an SCCS delta

SYNOPSIS

cdc −rSID [−**m**[mrlist]] [−**y**[comment]] files

DESCRIPTION

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

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

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

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

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

−r*SID* Used to specify the *S*CCS *ID*entification (*SID*) string of a delta for which the delta commentary is to be changed.

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

MR entries are added to the list of **MR**s in the same manner as that of *delta*(1). In order to delete an **MR**, precede the **MR** number with the character **!** (see *EXAMPLE*). If the **MR** to be deleted is currently in the list of **MR**s, it is removed and changed into a "comment" line. A list of all deleted **MR**s is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

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

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

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, *cdc* terminates and the delta commentary remains unchanged.

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

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

The exact permissions necessary to modify the SCCS file are documented in the **SCCS** chapter of the *Programming Tools Guide*. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLE

```
cdc −r1.6 −m"bl78-12345 !bl77-54321 bl79-00001" −ytrouble s.file
```

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

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

does the same thing.

WARNINGS

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

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
SCCS in the *Programming Tools Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cflow − generate C flowgraph

SYNOPSIS

cflow [**−r**] [**−ix**] [**−i_**] [**−dnum**] files

DESCRIPTION

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

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

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

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

The following options are interpreted by *cflow*:

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

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

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

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

EXAMPLE

Given the following in "file.c":

```
int     i;

main()
{
```

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

the command:

```
cflow −ix file.c
```

produces the output:

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

DIAGNOSTICS

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

SEE ALSO

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

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

CHECKCW (1) **SEE** *CW* **CHECKCW (1)**

CHECKEQ (1) **SEE** *EQN* **CHECKEQ (1)**

CHECKMM (1) **SEE** *MM* **CHECKMM (1)**

CHGRP (1) **SEE** *CHOWN* **CHGRP (1)**

NAME

chmod − change mode

SYNOPSIS

chmod mode files

DESCRIPTION

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

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

A symbolic *mode* has the form:

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

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

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

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

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

Only the owner of a file (or the super-user) may change its mode. Only the super-user may set the sticky bit. In order to set the group ID, the group of the file must correspond to your current group ID.

EXAMPLE

```
chmod 755 filename
```

changes the mode of "filename" to: read, write, execute (400+200+100) by owner; read, execute (40+10) for group; read, execute (4+1) for others. An *ls -l* of filename shows [-rwxr-xr-x filename] that the requested mode is in effect.

```
chmod = filename
```

will take away all permissions from *filename*, including yours.

```
chmod o-w file
```

denies write permission to others.

```
chmod +x file
```

makes a file executable.

SEE ALSO

ls(1), chmod(2).

NAME

chown, chgrp — change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

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

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

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

EXAMPLE

```
chown unisoft filea fileb filec
```

would make "unisoft" the owner of the three files.

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), chown(2), group(4), passwd(4).

NAME

clear — clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type (**TERM**) and capabilities string (**TERMCAP**). If **TERMCAP** is not found in the environment, it looks in **/etc/termcap** to figure out how to clear the screen.

EXAMPLE

```
clear
```

clears the screen.

FILES

/etc/termcap	terminal capability data base

SEE ALSO

environ(5), termcap(5)

NAME

cmp − compare two files

SYNOPSIS

cmp [−l] [−s] file1 file2

DESCRIPTION

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

Options:

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

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

EXAMPLE

```
cmp alpha beta
```

will report if the files are different and at what point they differ, such as:

```
alpha beta differ: char 33, line 2
```

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

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

NAME

col − filter reverse line-feeds

SYNOPSIS

col [**−bfpx**]

DESCRIPTION

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

If the **−b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

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

Unless the **−x** option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters SO (\016) and SI (\017) are assumed by *col* to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output SI and SO characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, SI, SO, VT (\013), and ESC followed by 7, 8, or 9. The VT character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, *col* will ignore any unknown escape sequences found in its input; the **−p** option may be used to cause *col* to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

EXAMPLE

```
nroff -ms filea | col
```

pipes multicolumn *nroff* output through the *col* filter to enable proper creation of columns.

SEE ALSO

nroff(1), tbl(1).

NOTES

The input format accepted by *col* matches the output produced by *nroff* with either the **−T37** or **−Tlp** options. Use **−T37** (and the **−f** option of *col*) if the ultimate disposition of the output of *col* will be a device that can interpret half-line motions, and **−Tlp** otherwise.

BUGS

Cannot back up more than 128 lines.

Allows at most 800 characters, including backspaces, on a line.

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

NAME

comb − combine SCCS deltas

SYNOPSIS

comb [−o] [−s] [−psid] [−clist] files

DESCRIPTION

Comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The generated shell procedure is written on the standard output.

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

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

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

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

−s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 ∗ (original − combined) / original

It is recommended that before any SCCS files are 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.

EXAMPLE

```
comb s.file1 > tmp1
```

produces a shell script saved in "tmp1" which will remove from the SCCS-format file, "s.file1", all deltas previous to the last set of changes, i.e., removes the capability to return to earlier versions.

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sh(1), sccsfile(4).
SCCS in the *Programming Tools Guide*.

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 − select or reject lines common to two sorted files

SYNOPSIS

comm [**−** [**123**]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name − 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.

EXAMPLE

```
comm -12 filea fileb
```

prints only the lines common to filea and fileb.

```
comm -23 filea fileb
```

prints only lines in the first file but not in the second.

```
comm -123 filea fileb
```

is not an option, as it suppresses all output.

```
comm -3 filea fileb
```

prints only the lines that differ in the two files.

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1).

NAME

conv – object file converter

SYNOPSIS

conv [–] [–a] [–o] [–p] [–s] –t *target files*

DESCRIPTION

The **conv** command converts object files from their current format to the format of the *target* machine. The converted file is written to file.v.

Flag options are:

–	read *files* from standard input.
–a	If the input file is an archive, produce the output file in the old archive format.
–o	If the input file is an archive, produce the output file in the UNIX 6.0 (Version 6) portable archive format.
–p	UNIX V.0 random access archive format. This is the default.
–s	Function exactly as 3bswab, i.e., preswab all characters in the object file. This is useful only for 3B20 object files which are to be swab-dumped from a DEC machine to a 3B20.
–t *target*	Convert the object file to the byte ordering of the machine (target) to which the object file is being shipped. This may be another host or a target machine. Legal values for *target* are: **pdp, vax, ibm, i86, x86, b16, n3b, m32,** and **m68k.**

conv can be used to convert all object files in common object file format. It can be used on either the source (sending) or target (receiving) machine.

conv is meant to ease the problems created by a multihost cross-compilation development environment. **conv** is best used within a procedure for shipping object files from one machine to another.

conv will recognize and produce archive files in three formats: the UNIX pre-V.0 format, the V.0 random access format, and the 6.0 portable ASCII.

EXAMPLE

```
*ship object files from pdp11 to ibm
$echo *.out | conv -t ibm
$uucp *.v my370!~/rje/
```

FILES

/bin/conv

DIAGNOSTICS

All diagnostics for the conv command are intended to be self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics on an input file cause the program to continue to the next input file.

WARNINGS

conv does not convert archives from one format to another if both the source and target machines have the same byte ordering.

NAME

cp – copy files

SYNOPSIS

cp [**–i**] [**–r**] *file1 file2*

cp [**–i**] [**–r**] *file ... directory*

DESCRIPTION

File1 is copied onto *file2*. The mode and owner of *file2* are preserved if it already existed; otherwise, the mode of the source file is used (all bits set in the current **umask** value are cleared).

In the second form, one or more *files* are copied into the *directory* with their original filenames.

Cp refuses to copy a file onto itself.

If the **–i** flag option is specified, *cp* will prompt the user with the name of the file whenever the copy will cause an old file to be overwritten. An answer of **y** will cause *cp* to continue. Any other answer will prevent it from overwriting the file.

If the **–r** flag option is specified and any of the source files are directories, *cp* copies each subtree rooted at that name; in this case, the destination must be a directory.

FILES

/bin/cp

SEE ALSO

cat(1), pr(1), mv(1), rcp(1N).

WARNING

Cp does not copy the description of special files, but attempts to copy the contents of the special file. This often occurs when using the **–r** flag option for a recursive copy. For example, *cp* will hang when trying to copy a named pipe or tty device. When a disk node is being copied, the contents of the disk partition will be copied. To copy the description of the special files, use *cpio*(1).

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio –o[acBv]

cpio –i[BcdmrtuvfsSb6] [patterns]

cpio –p[adlmruv] directory

DESCRIPTION

Cpio **–o** (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. The list of pathnames must contain only one file per line. (Thus, only some commands, such as *find* or *ls* without the –C option, will work in a pipeline to *cpio*.) Output is padded to a 512-byte boundary.

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

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

The meanings of the available options are:

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

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

d *Directories* are to be created as needed.

c Write *header* information in ASCII character form for portability.

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

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

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

- **v** *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls –l** command (see *ls*(1)).
- **l** Whenever possible, link files rather than copying them. Usable only with the **–i** or **–p** options.
- **m** Retain previous file modification time. This option is ineffective on directories that are being copied.
- **f** Copy in all files except those in *patterns*.
- **s** Swap bytes. Use only with the **–i** option.
- **S** Swap halfwords. Use only with the **–i** option.
- **b** Swap both bytes and halfwords. Use only with the –i option.
- **6** Process an old (i.e., UNIX System *Sixth* Edition format) file. Only useful with **–i** (copy in).

EXAMPLES

```
ls | cpio –o >/dev/mt/0m
```

copies the contents of a directory into an archive;

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

duplicates a directory hierarchy.

The trivial case "find . –depth –print | cpio –oB >/dev/rmt/0m" can be handled more efficiently by:

```
find . –cpio /dev/rmt/0m
```

SEE ALSO

ar(1), find(1), ls(1) cpio(4).

DIAGNOSTICS

Cpio complains and takes no action if current files are newer than those with the same name being extracted from an archive. Thus, newer files are never overwritten when extracting files from an archive with *cpio* **-i**.

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, thereafter, linking information is lost. Only the super-user can copy special files. The **–B** option does not work with certain magnetic tape drives.

NAME

cpp − the C language preprocessor

SYNOPSIS

/lib/cpp [option ...] [ifile [ofile]]

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. The output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc*(1) command since the functionality of *cpp* may someday be moved elsewhere. See *m4*(1) for a general macro processor.

Cpp optionally accepts two filenames as arguments. *Ifile* is the input and *ofile* is the output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

−P Preprocess the input without producing the line control information used by the next pass of the C compiler.

−C Pass along all comments except those found on *cpp* directive lines. By default, *cpp* strips C-style comments.

−U*name*

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, vax, m68k
UNIX System variant:	RES, RT

−D*name*
−D*name***=***def*

Define *name* as if by a **#define** directive. If no **=***def* is given, *name* is defined as 1.

−I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. When this option is used, **#include** files whose names are enclosed in "" are searched for first in the directory of the *ifile* argument, then in directories named in **−I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in **<>**, the directory of the *ifile* argument is not searched.

Two special names are understood by *cpp*. The name _LINE_ is defined as the current line number (as a decimal integer) as known by *cpp*, and _FILE_ is defined as the current filename (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines begun by **#**. The directives are:

#define *name token-string*

Replace subsequent instances of *name* with *token-string*.

NAME

cpp – the C language preprocessor

SYNOPSIS

/lib/cpp [**–C**] [**–D***name*[**=***def*]] [**–I***dir*] [**–P**] [**–U***name*] [*ifile*[*ofile*]]

DESCRIPTION

cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the **cc**(1) command. The output of **cpp** is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, **cpp** and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of **cpp** other than in this framework is not suggested. The preferred way to invoke **cpp** is through the **cc**(1) command since the functionality of **cpp** may someday be moved elsewhere. See **m4**(1) for a general macro processor.

cpp optionally accepts two filenames as arguments. *ifile* is the input and *ofile* is the output for the preprocessor. They default to standard input and standard output if not supplied.

The following flag options to **cpp** are recognized:

–P Preprocess the input without producing the line control information used by the next pass of the C compiler.

–C Pass along all comments except those found on **cpp** directive lines. By default, **cpp** strips C-style comments.

–U*name*

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, vax, m68k
UNIX System variant:	RES, RT

–D*name*

–D*name***=***def*

Define *name* as if by a **#define** directive. If no **=***def* is given, *name* is defined as 1.

–I*dir*

Search for **#include** files (whose names do not begin with **/**) in *dir* before looking in the directories on the standard list. When this flag option is used, **#include** files whose names are enclosed in **" "** are searched for first in the directory of the *ifile* argument, then in directories named in **–I** flag options, and last in directories on a standard list. For **#include** files whose names are enclosed in **<>**, the directory of the *ifile* argument is not searched.

Two special names are understood by **cpp**. The name **_LINE_** is defined as the current line number (as a decimal integer) as known by **cpp**, and **_FILE_** is defined as the current filename (as a C string) as known by **cpp**. They can be used anywhere (including in macros) just as any other defined name.

All **cpp** directives start with lines begun by **#**. The directives are:

#define *name token-string*

Replace subsequent instances of *name* with *token-string*.

#define *name*(*arg*, . . ., *arg*) *token-string*

Notice that there can be no space between *name* and the left parenthesis [(]. Replace subsequent instances of *name* followed by a left parenthesis [(], a list of comma-separated tokens, and a right parenthesis [)] by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma-separated list.

#undef *name*

Cause the definition of *name* (if any) to be forgotten from now on.

#include "*filename*"
#include <*filename*>

Include at this point the contents of *filename* (which will then be run through cpp). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **–I** flag option above for more detail.

#line *integer-constant* "*filename*"

Causes **cpp** to generate line control information for the next pass of the C compiler. *integer-constant* is the line number of the next line and *filename* is the file where it comes from. If "*filename*" is not given, the current filename is unchanged.

#endif

Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name*

The lines following appear in the output if and only if *name* has been the subject of a previous **#define** (or **–D**) without being the subject of an intervening **#undef**.

#ifndef *name*

The lines following do not appear in the output if and only if *name* has been the subject of a previous **#define** (or **–D**) without being the subject of an intervening **#undef**.

#if *constant-expression*

Lines following appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary –, !, and ˜ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by **cpp** should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else

Reverses the notion of the test directive that matches this directive. If lines previous to this directive are ignored, the following lines appear in the output. If lines previous to this directive are not ignored, the following lines do not appear in the output.

The test directives and the possible **#else** directives can be nested.

FILES

/lib/cpp
/usr/include

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

The error messages produced by **cpp** are self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When newline characters were found in argument lists for macros to be expanded, previous versions of **cpp** put out the newlines as they were found and expanded. The current version of **cpp** replaces these newlines with blanks to alleviate problems that the previous versions had when this occurred.

NAME

cpp − the C language preprocessor

SYNOPSIS

/lib/5.0/cpp [option ...] [ifile [ofile]]

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc5.0*(1) command. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc5.0*(1) command since the functionality of *cpp* may someday be moved elsewhere. See *m4*(1) for a general macro processor.

Cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

−P Preprocess the input without producing the line control information used by the next pass of the C compiler and leave the result in *ifile.o*.

−E Preprocess the input and put the resulting output on the standard output.

−C By default, *cpp* strips C-style comments. If the **−C** option is specified, all comments (except those found on *cpp* directive lines) are passed along.

−U*name*

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, m68000, pdp11, u370, u3b, vax
UNIX System variant:	RES, RT

−D*name*

−D*name***=***def*

Define *name* as if by a **#define** directive. If no *=def* is given, *name* is defined as 1.

−I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in "" will be searched for first in the directory of the *ifile* argument, then in directories named in **−I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in **<>**, the directory of the *ifile* argument is not searched.

Two special names are understood by *cpp*. The name **__LINE__** is defined as the current line number (as a decimal integer) as known by *cpp*, and **__FILE__** is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines begun by **#**. The directives are:

#define *name token-string*
: Replace subsequent instances of *name* with *token-string*.

#define *name(arg, ..., arg) token-string*
: Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list.

#undef *name*
: Cause the definition of *name* (if any) to be forgotten from now on.

#include "*filename*"
#include <*filename*>
: Include at this point the contents of *filename* (which will then be run through *cpp*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **−I** option above for more detail.

#line *integer-constant "filename"*
: Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If "*filename*" is not given, the current file name is unchanged.

#endif
: Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name*
: The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name*
: The lines following will not appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*
: Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary −, !, and ˜ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else
: Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output. And vice versa.

The test directives and the possible **#else** directives can be nested.

EXAMPLE

```
/lib/cpp  −P  −DXYZ  −DMYFILE=myfile  −I../include  myprog.c
myprog.i
```

would preprocess "myprog.c" input output file "myprog.i", deleting output line numbers (−**P**), defining symbol XYZ to be null, symbol MYFILE to be "myfile" and using include files from ../**include**.

FILES

/usr/include standard directory for **#include** files

SEE ALSO

cc5.0(1), m4(1).

DIAGNOSTICS

The error messages produced by *cpp* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When newline characters were found in argument lists for macros to be expanded, previous versions of *cpp* put out the newlines as they were found and expanded. The current version of *cpp* replaces these newlines with blanks to alleviate problems that the previous versions had when this occurred.

NAME

crontab – user crontab file

SYNOPSIS

crontab [file]
crontab –r
crontab –l

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The –r option removes a user's crontab from the crontab directory. *Crontab* –l will list the crontab file for the invoking user.

A user is permitted to use *crontab* if their name appears in the file **/usr/lib/cron/cron.allow**. If that file does not exist, the file **/usr/lib/cron/cron.deny** is checked to determine if the user should be denied access to **crontab**. If neither file exists, only root is allowed to submit a job. The allow/deny files consist of one user name per line.

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

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

Each of these patterns may be either an asterisk (meaning all legal values), or a list of elements separated by commas. An element is either a number, or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh.** Users who desire to have their *.profile* executed must do so in the crontab file explicitly by prepending the statement:

```
. .profile &&
```

to the command being run. This must be specified for each line of the **crontab** for which the *.profile* is to be sourced. *Cron* supplies a default environment for every shell, defining **HOME, LOGNAME, SHELL(=/bin/sh),** and **PATH(=:/bin:/usr/bin:/usr/lbin).**

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

sh(1), cron(1M).

NAME

crypt − encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; ''sneak paths'' by which keys or clear text can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e., to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lowercase letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of *crypt*.

EXAMPLE

```
crypt asa < sleeper.c > zzz
```

will use the string "asa" as key to the encryption algorithm to encrypt the contents of "sleeper.c", and place the encrypted output in file "zzz". File "zzz" at this point will be unreadable. NOTE that the original file, "sleeper.c", remains in readable form. To obtain readable print-out of the file "zzz", it could be decoded as follows:

```
crypt < zzz
```

After the response:

```
Enter key:
```

the user types in "asa".

FILES

/dev/tty   for typed key

strings separated by newlines, as for *fgrep* (1).

When the shell's input is not a terminal, the character # introduces a comment which continues to the end of the input line. It is prevented from having this special meaning when preceded by \ or if bracketed by a pair of single or double quotation marks.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed.

A simple command or a sequence of simple commands separated by | characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next.

Sequences of pipelines may be separated by ;, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an &, which means "run it in background".

Parentheses (and) around a pipeline or sequence of pipelines cause the whole series to be treated as a simple command, which may in turn be a component of a pipeline, etc. It is also possible to separate pipelines with || or && indicating, as in the C language, that the second is to be executed only if the first fails or succeeds, respectively. (See *Expressions*.)

Process ID Numbers

When a process is run in background with &, the shell prints a line which looks like:

```
1234
```

indicating that the process which was started asynchronously was number 1234.

Status Reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

To check on the status of a process, use the *ps* (process status) command.

Substitutions

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

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence.

History substitutions begin with the character ! and may begin **anywhere** in the input stream (with the proviso that they **do not** nest.)

This ! may be preceded by a \ to turn off its special meaning; for convenience, a ! is also passed unchanged when it is followed by a blank, tab, newline, = or (.

Therefore, **do not** put a space after the ! and the command reference when you are invoking the shell's history mechanism. (History substitutions also occur when an input line begins with ↑. This special abbreviation will be described later.)

An input line which invokes history substitution is echoed on the terminal before it is executed, as it would look if typed out in full.

The shell's history list, which may be seen by typing the *history* command, contains all commands input from the terminal which consist of one or more words. History substitutions reintroduce sequences of words from these saved commands into the input stream. The *history* variable controls the size of the input stream. The previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

Consider the following output from the *history* command:

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

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an ! in the prompt string. This is done by SETting Prompt = ! and the prompt character of your choice.

For example, if the current event is number 13, we can call up the command recorded as event 11 in several ways: **!-2** [i.e., 13-2]; by the first letter of one of its command words, such as **!c** referring to the "c" in *cat*; or **!wri** for event 9, or by a string contained in a word in the command as in **!?mic?** also referring to

SEE ALSO

ed(1), makekey(1), stty(1).

BUGS

If output is piped to *nroff* and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)).
If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

NOTE

This utility is not provided with international distribution.

NAME

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

SYNOPSIS

csh [–cefinstvVxX] [arg ...]

DESCRIPTION

Csh is a command language interpreter incorporating a history mechanism (see **History Substitutions**) and a C-like syntax.

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

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

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates, it executes commands from the file ".logout" in the user's home directory.

Lexical Structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters &, |, ;, <, >, (,), form separate words. If doubled in &&, ||, << or >>, these pairs form single words. These parser metacharacters may be made part of other words, or their special meaning may be prevented, by preceding them with a backslash (\). A newline preceded by a \ is equivalent to a blank. It is usually necessary to use the backslash to *escape* the parser metacharacters when you want to use them literally rather than as metacharacters.

Strings enclosed in matched pairs of quotation marks, either single or double quotation marks, ', ‘ or ", form parts of a word. Metacharacters in these strings, including blanks and tabs, do not form separate words. Such quotations have semantics to be described subsequently.

Within pairs of single or double quotation marks, a newline (carriage return) preceded by a \ gives a true newline character. This is used to set up a file of event 9.

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case !! refers to the previous command; thus !! alone is essentially a *redo*.

Words are selected from a command event and acted upon according to the following formula:

event:position:action

The *event* is the command you wish to retrieve. As mentioned above, it may be summoned up by event number and in several other ways. All that the *event* notation does is to tell the shell which command you have in mind.

Position picks out the words from the command event on which you want the *action* to take place. The *position* notation can do anything from altering the command completely to making some very minor substitution, depending on which words from the command event you specify with the *position* notation.

To select words from a command event, follow the event specification with a : and a designator (by position) for the desired words.

The words of a command event are picked out by their position in the input line. Positions are numbered from 0, the first word (usually command) being position 0, the second word having position 1, and so forth. If you designate a word from the command event by stating its position, means you want to include it in your revised command. All the words that you want to include in a revised command must be designated by position notation in order to be included.

The basic position designators are:

0	first (command) word
n	*n*th argument
↑	first argument, i.e., 1
$	last argument
%	matches the word of an ?s? search which immediately precedes it; used to strip one word out of a command event for use in another command. Example: **!?four?:%:p** prints **four**.
x–*y*	range of words (e.g., 1-3 means "from position 1 to position 3").
–*y*	abbreviates "0–*y*
*	stands for "↑–$", or indicates position 1 if only one word in event.

*x** abbreviates "*x*−$" where *x* is a position number.

x− like "*x**" but omitting last word "$"

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ↑, $, *, − or %.

Modifiers, each preceded by a :, may be used to act on the designated words in the specified command event. The following modifiers are defined:

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

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

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

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

You can quickly make substitutions to the previous command line by using the ↑ character as the first non-blank character of an input line. This is equivalent to **!:s↑** providing a convenient shorthand for substitutions on the text of the previous line. Thus **↑lb↑lib** fixes the spelling of "lib" in the previous command.

Finally, a history substitution may be surrounded with { and } if necessary to insulate it from the characters which follow. Thus, after **ls –ld ˜paul** we might do **!{l}a** to do **ls –ld "paula**, while **!la** would look for a command starting **la**.

Quotations with ' and "

The quotation of strings by ' and " can be used to prevent all or some of the remaining substitutions which would otherwise take place if these characters were interpreted as "metacharacters" or "wild card matching characters". Strings enclosed in single quotes, ' are prevented any further interpretation or expansion. Strings enclosed in " may still be variable and command expanded as described below.

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

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

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

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

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

Variable substitution

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

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

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

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by $ characters. This expansion can be prevented by preceding the $ with a \ except within double quotes (") where it **always** occurs, and within single quotes (') where it **never** occurs. Strings quoted by ` are interpreted later (see *Command substitution* below) so $ substitution does not occur there until later, if at all. A $ is passed unchanged if followed by a blank, tab, or end-of-line.

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

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

Metasequences for variable substitution

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

$name
${name}

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

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

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

$#name
${#name}

Gives the number of words in the variable. This is useful for later use in a "[selector]".

$0

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

$number
${number}

Equivalent to "$argv[number]".

$*

Equivalent to "$argv[*]".

The modifiers ":h", ":t", ":r", ":q" and ":x" may be applied to the substitutions above as may ":gh", ":gt" and ":gr". If braces { } appear in the command form, then the modifiers must appear within the braces. The current implementation allows only one : modifier on each $ expansion.

The following substitutions may not be modified with : modifiers.

$?name
${?name}

Substitutes the string "1" if name is set, "0" if it is not.

$?0

Substitutes "1" if the current input filename is known, "0" if it is not.

$$

Substitute the (decimal) process number of the (parent) shell.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

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

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

Filename substitution

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

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

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

The metanotation **a{b,c,d}e** is a shorthand for **abeaceade**. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus **˜source/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/ls.c** whether or not these files exist without any chance of error if the home directory for **source** is **/usr/source**. Similarly **../{memo,*box}** might expand to **../memo ../box ../mbox**. (Note that "memo" was not sorted with the results of matching "*box".) As a special case {, } and { } are passed undisturbed.

Input/output

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

< name

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

<< word

Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, ", ´ or ` appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote $, \

and `. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name

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

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

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

>> name
>>& name
>>! name
>>&! name

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

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

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

Expressions

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

|| && | ↑ & == != =~ !~ <= >= < > << >> + − * / % ! ~ ( )

Here the precedence increases to the right, ==, !=, =~ and !~; <=, >=, < and >; << and >>; + and −; *, / and % being, in groups, at the same level. The ==, !=, =~ and !~ operators compare their arguments as strings; all others operate on numbers. The operators =~ and !~ are like != and == except that the right hand side is a *pattern* (containing, e.g., *s, ?s and instances of [...]) against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

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

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

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

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

Control Flow

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

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

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward *goto* s will succeed on non-seekable inputs.)

Builtin Commands

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

alias
alias name
alias name wordlist

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

break

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

breaksw

> Causes a break from a *switch*, resuming after the *endsw*.

case label:

> A label in a *switch* statement as discussed below.

cd
cd name

chdir
chdir name

Change the shells working directory to directory *name*. If no argument is given, then change to the home directory of the user.

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

continue

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

default:

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

echo wordlist
echo −n wordlist

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

else
end
endif
endsw

See the description of the *foreach*, *if*, *switch*, and *while* statements below.

exec command

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

exit
exit(expr)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)
...
end

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

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with ? before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal, you can rub it out.

glob wordlist

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

goto word

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

history

Displays the history event list.

if (expr) command

If the specified expression evaluates true, then the single *command* with arguments is executed. *Command* must be a simple command, not a pipeine, a command list, or a parenthesized command list. In the interactive shell, the *if* statement can accept only one simple command after the *expr* and on the same line as the *expr* . Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif

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

kill pid
kill −sig pid ...

Sends either the TERM (terminate) signal or specified signal to the specified processes. Signals are either given by number or names (as in **/usr/include/signal.h**, stripped of the prefix SIG). There is no default, saying "kill" does not send a signal to the current process. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

login

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

logout

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

newgrp

changes the group identification of its caller resulting in the access permissions being calculated with respect to the new group ID.

nice
nice +number
nice command
nice +number command

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

nohup
nohup command

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

onintr
onintr −
onintr label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form **onintr**

– causes all interrupts to be ignored. The final form causes the shell to execute a **goto label** when an interrupt is received or a child process terminates because it was interrupted.

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

rehash

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

repeat count command

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

set
set name
set name=word
set name[index]=word
set name=(wordlist)

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

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

setenv name value

Sets the value of environment variable *name* to be *value*, a single string. The variable PATH is automatically imported to and exported from the *csh* variable *path*; there is no need to use *setenv* for these.

shift
shift variable

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

source name

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

```
switch (string)
case str1:
   ...
    breaksw
...
default:
   ...
    breaksw
endsw
```

Each case label is successively matched against the specified *string* which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a "default" label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command

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

umask

umask value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern

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

unhash

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

unset pattern

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

wait

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

while (expr)
...
end

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

@
@ name = expr
@ name[index] = expr

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

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

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

Pre-defined and Environment Variables

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

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file ".cshrc" as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it.

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

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

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

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

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

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

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

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

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

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

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

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

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

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

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

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status "1", all other builtin commands set status "0".

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

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

Non-builtin Command Execution

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

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

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

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

Argument List Processing

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

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

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

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

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

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

–s Command input is taken from the standard input.

–t A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.

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

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

–V Causes the *verbose* variable to be set even before ".cshrc" is executed.

–X Is to **–x** as **–V** is to **–v**.

After processing of flag arguments, if arguments remain but none of the **–c**, **–i**, **–s**, or **–t** options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by "$0". Remaining arguments initialize the variable *argv*.

Signal Handling

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

EXAMPLE

```
csh
```

creates a new shell which will accept shell commands with Berkeley extensions.

FILES

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

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 5120 characters. The number of arguments to a command which involves filename expansion is limited to 1/6th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), a.out(4), environ(5)
CSH in the *User Guide*.

BUGS

It suffices to place the sequence of commands in ()s to force it to a subshell, i.e., "(a ; b ; c)".

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

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

When an attempt is made to redirect standard error messages during a source command (as in "source *command* >& *filename*)), the shell creates *filename* but sends both standard output and standard error to the terminal, not the file.

Commands within loops, prompted for by ?, are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

Symbolic links confuse the shell. In particular, dirs and **cd ..** don't work properly once you've crossed through a symbolic link.

AUTHOR

William Joy.

NAME

csplit − context split

SYNOPSIS

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

DESCRIPTION

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

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

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

⋮

n+1: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a − then standard input is used.

The options to *csplit* are:

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

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

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

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

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

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

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

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

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *Csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLE

```
csplit −f cobol file '/procedure division/' /par5./ /par16./
```

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

```
cat cobol0[0−3] > file
```

Note that this example overwrites the original file.

```
csplit −k file  100  {99}
```

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

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

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

SEE ALSO

ed(1), sh(1), regexp(5).

DIAGNOSTICS

Self explanatory except for:

```
arg − out of range
```

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

NAME

ct − spawn getty to a remote terminal

SYNOPSIS

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

DESCRIPTION

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

Ct will try each line listed in the file **/usr/lib/uucp/L-devices** until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, *ct* will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. *Ct* will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the **−w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

Normally, *ct* will hang up the current line, so that that line can answer the incoming call. The **−h** option will prevent this action. If the **−v** option is used, *ct* will send a running narrative to the standard error output stream.

The data rate may be set with the **−s** option, where *speed* is expressed in baud. The default rate is 300.

After the user on the destination terminal logs out, *ct* prompts, **Reconnect?** If the response begins with the letter **n** the line will be dropped; otherwise, *getty* will be started again and the **login:** prompt will be printed.

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

EXAMPLE

```
ct -w15 -s1200 644-1234
```

dials from the terminal the given modem phone number (644-1234), spawning a login process at 1200 baud. If the dialer line is busy, *ct* will continue to try to open the dialer at one-minute intervals for a total of 15 minutes (as set by the **−w** option).

FILES

/usr/lib/uucp/L-devices
/usr/adm/ctlog

SEE ALSO

cu(1C), login(1), uucp(1C).

NAME

ctags – maintain a tags file for a C program

SYNOPSIS

ctags [**–a**] [**–u**] [**–w**] [**–x**] name ...

DESCRIPTION

Ctags makes a tags file for *ex*(1) and *vi*(1) from the specified C, Fortran, and Pascal sources.

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

Options

The **–a** option causes the output to be appended to the tags file instead of rewriting it.

The **–u** option causes the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

The **–w** option suppresses warning diagnostics.

If the **–x** flag is given, *ctags* produces a list of function names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output.

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

The tag *main* is treated specially in C programs. The tag formed is created by prepending "M" to the name of the file, with a trailing ".c" removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

EXAMPLE

```
ctags *.c *.h
```

puts the tags from all the ".c" and ".h" files into the tagsfile "tags".

FILES

tags	output tags file

SEE ALSO

ex(1), vi(1).

BUGS

Not all warning diagnostics are suppressed by −w.

If *ctags* is interrupted while executing under the −u option, a temporary file named OTAGS is left in the current directory.

AUTHOR

Ken Arnold

NAME

ctrace − C program debugger

SYNOPSIS

ctrace [options] [file]

DESCRIPTION

Ctrace allows you to follow the execution of a C program, statement by statement. The effect is similar to executing a shell procedure with the **−x** option. *Ctrace* reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc*(1) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(1) or *tail*(1) commands.

The only options you will commonly use are:

−f *functions*	Trace only these functions.
−v *functions*	Trace all but these functions.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:

−o	Octal
−x	Hexadecimal
−u	Unsigned
−e	Floating point

These options are used only in special circumstances:

−l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

−s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

−t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.

−P Run the C preprocessor on the input before tracing it. You can also use the **−D**, **−I**, and **−U** *cc*(1) preprocessor options.

These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX system environment:

−b Use only basic functions in the trace code, that is, those in *ctype*(3C), *printf*(3S), and *string*(3C). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(2), *fflush*(3S), *longjmp*(3C), or *setjmp*(3C).

−p *'s'* Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.

−r *f* Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **−p** option).

EXAMPLE

If the file *lc.c* contains this C program:

```
1 #include <stdio.h>
2 main()        /* count lines in input */
3 {
4       int c, nl;
5
6       nl = 0;
7       while ((c = getchar()) != EOF)
8               if (c = '\n')
9                       ++nl;
10      printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
(ctrl-d),
```

the program will be compiled and executed. The output of the program will be the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
 2 main()
 6      nl = 0;
```

```
        /* nl == 0 */
7       while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
        /* c == 49 or '1' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 1 */
7       while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 2 */
7       while ((c = getchar()) != EOF)
```

If you now enter an end of file character (ctrl-d) the final output will be:

```
        /* c == -1 */
10      printf("%d\n", nl);
        /* nl == 2 */2
         return
```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equal operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the **−f** or **−v** options to trace specific functions. This does not give you statement by statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding *ctroff*() and *ctron*() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from *sdb*(1) if you compile with the **−g** option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
r
```

You can also turn the trace off and on by setting static variable *tr_ct_* to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly

DIAGNOSTICS

This section contains diagnostic messages from both *ctrace* and *cc*(1), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

Ctrace Diagnostics

warning: some variables are not traced in this statement

Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the **−t** option to increase this number.

warning: statement too long to trace

This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use −P option

This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

'if ... else if' sequence too long

Split the sequence by removing an else from the middle.

possible syntax error, try −P option

Use the -P option to preprocess the *ctrace* input, along with any appropriate **−D**, **−I**, and **−U** preprocessor options. If you still get the error message, check the Warnings section below.

Cc Diagnostics

warning: floating point not implemented
warning: illegal combination of pointer and integer
warning: statement not reached
warning: sizeof returns 0

Ignore these messages.

compiler takes size of function

See the *ctrace* "possible syntax error" message above.

yacc stack overflow

See the *ctrace* "'if ... else if' sequence too long" message above.

out of tree space; simplify expression

Use the **−t** option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

redeclaration of signal

Either correct this declaration of *signal*(2), or remove it and #include <signal.h>.

WARNINGS

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

Ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g., "int EOF;", will cause a syntax error.

BUGS

Ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. Ctrace may choose to print the address of an aggregate or use the wrong format (e.g., %e for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES

runtime.c	run-time trace package

SEE ALSO

signal(2), ctype(3C), fflush(3S), longjmp(3C), printf(3S), setjmp(3C), string(3C).

NAME

cu − call another UNIX system

SYNOPSIS

cu [−sspeed] [−lline] [−h] [−t] [−d] [−m] [−o] [−e] [−n] telno | systemname | dir

DESCRIPTION

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

cu accepts the following options and arguments.

−sspeed
Specifies the transmission speed (110, 150, 300, 600, 1200, 4800, 9600); 300 is the default value. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

−lline Specifies a device name to use as the communication line. This can be used to override searching for the first available line having the right speed. When the −l option is used without the −s option, the speed of a line is taken from the file **/usr/lib/uucp/L-devices.** When the −l and −s options are used simultaneously, cu will search the L-devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/ttyab**), in this case a phone number is not required but the string dir may be use to specify a null acu. If the specified device is associated with an auto dialer, a phone number must be provided.

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

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

−d Causes diagnostic traces to be printed.

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

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

−m Designates a direct line which has modem control.

−n Will request the phone number to be dialed from the user rather than taking it from the command line.

telno When using an automatic dialer the argument is the telephone number with equal signs for secondary dial tone or minus signs for delays, at appropriate places.

systemname
A **uucp** system name may be used rather than a phone number; in this case, cu will obtain an appropriate direct line or phone number

from **/usr/lib/uucp/L.sys** (the appropriate baud rate is also read along with phone numbers). Cu will try each phone number or direct line for **systemname** in the L.sys file until a connection is made or all the entries are tried.

dir Using **dir** insures that cu will use the line specified by the −l option.

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

The *transmit* process interprets the following:

~. terminate the conversation.

~! escape to an interactive shell on the local system.

~!*cmd*... run *cmd* on the local system (via **sh −c**).

~$*cmd*... run *cmd* locally and send its output to the remote system.

~**%cd** change the directory on the local system. **NOTE:** ~**!cd** will cause the command to be run by a sub-shell; probably not what was intended.

~**%take** *from* [*to*] copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.

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

~~ ... send the line ~ ... to the remote system.

~**%break** transmit a **BREAK** to the remote system.

~**%nostop** toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

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

Data from the remote is diverted (or appended, if >> is used) to *file*. The trailing ~> terminates the diversion.

The use of ~**%put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of **~%take** requires the existence of *echo*(1) and *cat*(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

When **cu** is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. For example, uname can be executed on Z, X, and Y as follows:

```
uname
Z
~!uname
X
~~!uname
Y
```

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

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud:

```
cu -s1200  9=2015551212
```

If the speed is not specified, 300 is the default value.

To login to a system connected by a direct line:

```
cu -l /dev/ttyXX  dir
```

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

```
cu -s1200 -l /dev/ttyXX  dir
```

To dial a system using a specific line:

```
cu -l /dev/culXX  2015551212
```

To use a system name:

```
cu  YYYZZZ
```

FILES

/usr/lib/uucp/L.sys
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uname(1), uucp(1C).

DIAGNOSTICS

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

BUGS

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

NOTE

Any input character after an ~ will be preceded by [sysname] (inserted by cu).

NAME

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

SYNOPSIS

cut **−c**list [file1 file2 ...]

cut **−f**list [**−d**char] [**−s**] [file1 file2 ...]

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (**−c** option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (**−f** option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

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

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

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

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

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

Either the **−c** or **−f** option must be specified.

HINTS

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

EXAMPLE

```
cut −d: −f1,5 /etc/passwd
```

mapping of user IDs to names.

```
name=`who am i | cut −f1 −d" "`
```

to set *name* to current login name.

DIAGNOSTICS

line too long

A line can have no more than 1023 characters or fields.

bad list for c/f option

Missing **−c** or **−f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

no fields
The *list* is empty.

SEE ALSO

grep(1), **paste**(1).

NAME

cw, checkcw − prepare constant-width text for troff

SYNOPSIS

cw [−lxx] [−rxx] [−fn] [−t] [+t] [−d] [files]

checkcw [−lxx] [−rxx] files

DESCRIPTION

Cw is a preprocessor for *troff*(1) input files that contain t xt to be typeset in the constant-width (CW) font.

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. (An earlier version of this font was used in typesetting *The C Programming Language* by B. W. Kernighan and D. M. Ritchie.) It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

Because the CW font contains a "non-standard" set of characters and because text typeset with it requires different character and inter-word spacing than is used for "standard" fonts, documents that use the CW font must be preprocessed by *cw*.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$%&()`'*+@.,/:;=?[]|-_^~"<>{}#\
```

plus eight non-ASCII characters represented by four-character *troff*(1) names (in some cases attaching these names to "non-standard" graphics):

Character	*Symbol*	*Troff Name*
"Cents" sign	¢	\(ct
EBCDIC "not" sign	¬	\(no
Left arrow	←	\(<-
Right arrow	→	\(->
Down arrow	↓	\(da
Vertical single quote	'	\(fm
Control-shift indicator	†	\(dg
Visible space indicator	□	\(sq
Hyphen	-	\(hy

The hyphen is a synonym for the unadorned minus sign (-). Certain versions of *cw* recognize two additional names: \(ua for an up arrow and **\(lh** for a diagonal left-up (home) arrow.

Cw recognizes five request lines, as well as user-defined delimiters. The request lines look like *troff*(1) macro requests, and are copied in their entirety by *cw* onto its output; thus, they can be defined *by the user* as *troff*(1) macros; in fact, the .CW and .CN macros *should* be so defined (see *HINTS* below). The five requests are:

.CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the *cw* command line.

.CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the *cw* command line.

.CD Change delimiters and/or settings of other options; takes the same options as are available on the *cw* command line.

.CP *arg1 arg2 arg3 . . . argn*
All the arguments (which are delimited like *troff*(1) macro arguments) are concatenated, with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

.PC *arg1 arg2 arg3 . . . argn*
Same as .CP, except that the even-numbered arguments are set in the CW font and the odd-numbered ones in the prevailing font.

The .CW and .CN requests are meant to bracket text (e.g., a program fragment) that is to be typeset in the CW font "as is." Normally, *cw* operates in the *transparent* mode. In that mode, except for the .CD request and the nine special four-character names listed in the table above, every character between .CW and .CN request lines stands for itself. In particular, *cw* arranges for periods (.) and apostrophes (') at the beginning of lines, and backslashes (\) everywhere to be "hidden" from *troff*(1). The transparent mode can be turned off (see below), in which case normal *troff*(1) rules apply; in particular, lines that begin with . and ' are passed through untouched (except if they contain delimiters—see below). In either case, *cw* hides the effect of the font changes generated by the .CW and .CN requests; *cw* also defeats all ligatures (*fi*, *ff*, etc.) in the CW font.

The only purpose of the .CD request is to allow the changing of various options other than just at the beginning of a document.

The user can also define *delimiters*. The left and right delimiters perform the same function as the .CW/.CN requests; they are meant, however, to enclose CW "words" or "phrases" in running text (see example under *BUGS* below). *Cw* treats text between delimiters in the same manner as text enclosed by .CW/.CN pairs, except that, for aesthetic reasons, spaces and backspaces inside .CW/.CN pairs have the same width as other CW characters, while spaces and backspaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text (but are *not* adjustable). Font changes due to delimiters are *not* hidden.

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

−l*xx* The one- or two-character string *xx* becomes the left delimiter; if *xx* is omitted, the left delimiter becomes undefined, which it is initially.

−r*xx* Same for the right delimiter. The left and right delimiters may (but need not) be different.

−f*n* The CW font is mounted in font position *n*; acceptable values for *n* are 1, 2, and 3 (default is 3, replacing the bold font). This option is only useful at the beginning of a document.

−t Turn transparent mode *off*.

+t Turn transparent mode *on* (this is the initial default).

−d Print current option settings on file descriptor 2 in the form of *troff*(1) comment lines. This option is meant for debugging.

Cw reads the standard input when no *files* are specified (or when − is specified as the last argument), so it can be used as a filter. Typical usage is:

```
cw files | troff ...
```

Checkcw checks that left and right delimiters, as well as the .CW/.CN pairs, are properly balanced. It prints out all offending lines.

HINTS

Typical definitions of the .CW and .CN macros meant to be used with the *mm*(1) macro package:

```
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ...
..
.de CN
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *troff*(1) no-fill (.nf) mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point *smaller* than the prevailing point size (the displayed definitions of .CW and .CN above are one point smaller than the running text on this page). The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: *cw*, *tbl*, and *eqn*. Usually, the tables contained in such documents will not contain any CW text, although it is entirely possible to have *elements* of the table set in the CW font; of course, care must be taken that *tbl*(1) format information not be modified by *cw*. Attempts to set equations in the CW font are not likely to be either pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting ← represent a backspace, d←←† yields đ. Because spaces (and, therefore backspaces) are half as wide between delimiters as inside .CW/.CN pairs (see above), two backspaces are required for each overstrike between delimiters.

EXAMPLE

```
cw text | tbl | troff −mm
```

processes the text file "text", sends the output to *tbl*(1) and then sends the output for final formatting to *troff*(1) and *mm*(1).

FILES

/usr/lib/font/ftCW   CW font-width table

SEE ALSO

eqn(1), mmt(1), tbl(1), troff(1), mm(5), mv(5).

WARNINGS

If text preprocessed by *cw* is to make any sense, it must be set on a typesetter equipped with the CW font or on a STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

BUGS

Only a masochist would use periods (.), backslashes (\), or double quotes (") as delimiters, or as arguments to .CP and .PC.

Certain CW characters don't concatenate gracefully with certain Times Roman characters, e.g., a CW ampersand (&) followed by a Times Roman comma(,); in such cases, judicious use of *troff*(1) half- and quarter-spaces (\| and \^) is most salutary, e.g., one should use _&_\^, (rather than just plain _&_,) to obtain &, (assuming that _ is used for both delimiters).

Using *cw* with *nroff* is silly.

The output of *cw* is hard to read.

See also *BUGS* under *troff*(1).

NAME

cxref − generate C program cross-reference

SYNOPSIS

cxref [options] files

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross-reference table. *Cxref* utilizes a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **−c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **−D**, **−I** and **−U** options (which are identical to their interpretation by *cc*(1)), the following *options* are interpreted by *cxref*:

−c Print a combined cross-reference of all input files.

−w<*num*>
Width option which formats output no wider than <num> (decimal) columns. This option will default to 80 if <num> is not specified or is less than 51.

−o *file* Direct output to named *file*.

−s Operate silently; does not print input file names.

−t Format listing for 80-column width.

FILES

/usr/lib/xcpp special version of C-preprocessor.

SEE ALSO

cc(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you can't compile these files, anyway.

BUGS

Cxref considers a formal argument in a #*define* macro definition to be a declaration of that symbol. For example, a program that #*include*s **ctype.h**, will contain many declarations of the variable **c**.

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. (If a number of less than 70 is given for the year, the year that results is 1970.) 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*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by **%** and will be replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year – 01 to 12
d	day of month – 01 to 31
y	last 2 digits of year – 00 to 99
D	date as mm/dd/yy
H	hour – 00 to 23
M	minute – 00 to 59
S	second – 00 to 59
T	time as HH:MM:SS
j	day of year – 001 to 366
w	day of week – Sunday = 0
a	abbreviated weekday – Sun to Sat

h	abbreviated month – Jan to Dec
r	time in AM/PM notation

EXAMPLE

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

generates as output:

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

DIAGNOSTICS

No permission	if you are not the super-user and you try to change the date;
bad conversion	if the date set is syntactically incorrect;
bad format character	if the field descriptor is not recognizable.

SEE ALSO

printf(3S).

WARNING

It is a bad practice to change the date while the system is running multi-user.

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.

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

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

d The top value on the stack is duplicated.

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

f All values on the stack are printed.

q exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value. Alternately, control-d (EOF) will exit from *dc*.

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

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

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

<*x* **>***x* **=***x*

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

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

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

c All values on the stack are popped.

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

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

O pushes the output base on the top of the stack.

k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

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

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

; : are used by *bc* for array operations.

EXAMPLE

```
dc
24.2 56.2 + p
```

adds the two numbers and prints the result (top value in the stack).

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

prints the first ten values of *n*!.

SEE ALSO

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

DIAGNOSTICS

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

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

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

Out of headers for too many numbers being kept around.

Out of pushdown for too many items on the stack.

Nesting Depth for too many levels of nested execution.

NAME

dd − convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

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

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying; *dd* creates the specified output file (see *creat*(2)), which insures the length of the file will be zero for regular files; seeking *n* blocks from the beginning of the output file will fill the skipped area with zeros (nulls).
count=*n*	copy only *n* input blocks
conv=**ascii**	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
..., **...**	several comma-separated conversions
multi=**in**	input file is multi-volume
out	output file is multi-volume
in,out	both the input file and output file are multi-volume

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*, *ebcdic*, or *ibm* 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 two cases ASCII characters are read into the conversion buffer, converted to EBCDIC (or the IBM version of EBCDIC), and blanks added to make up an output block of size *cbs*.

If multi-volume input(output) is specified, a prompt is given on end-of-file to allow another volume to be mounted.

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

EXAMPLE

```
dd if=/dev/rmt/0m of=x ibs=800 cbs=80 conv=ascii,lcase
```

will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the ASCII file "x".

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 block sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

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

BUGS

The ASCII/ EBCDIC conversion tables are taken from the 256-character standard in the CACM Nov, 1968. The *ibm* conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

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

NAME

delta − make a delta (change) to an SCCS file

SYNOPSIS

delta [**−r**SID] [**−s**] [**−n**] [**−g**list] [**−m**[mrlist]] [**−y**[comment]] [**−p**] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see **−m** and **−y** keyletters below).

Keyletter arguments apply independently to each named file.

−r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *gets* for editing (**get −e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **−r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

−s Suppresses the issue on the standard output of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

−n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

−g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

−m[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (**MR**) number *must* be supplied as the reason for creating the new delta.

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

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

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from **MR** number validation program, *delta* terminates (it is assumed that the **MR** numbers were not all valid).

−y[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment.* If the comment includes spaces, you must enclose the entire string in double quotes.

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

−p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

EXAMPLE

```
% delta s.test1.c
 comments?  second version
 1.2
 1 inserted
 0 deleted
 12 unchanged
```

does a *delta* on file "test1.c".

FILES

All files of the form *?*-file are explained in the **SCCS** section of the *Programming Tools Guide*. The naming convention for these files is also described there.

g-file Existed before the execution of *delta*; removed after completion of *delta*.

p-file Existed before the execution of *delta*; may exist after completion of *delta*.

q-file Created during the execution of *delta*; removed after completion of *delta*.

x-file Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.

z-file Created during the execution of *delta*; removed during the execution of *delta*.

d-file Created during the execution of *delta*; removed after completion of *delta*.

/usr/bin/bdiff Program to compute differences between the "gotten" file and the *g-file*.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and will cause an error.

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

If the standard input (−) is specified on the *delta* command line, the **−m** (if necessary) and **−y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4).
SCCS in the *Programming Tools Guide*.

DIAGNOSTICS

Use *help* (1) for explanations.

NAME

deroff − remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [−mx] [−w] [files]

DESCRIPTION

Deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between .EQ and .EN lines, and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *Deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **−m** option may be followed by an **m**, **s**, or **l**. The **−mm** option causes the macros be interpreted so that only running text is output (i.e., no text from macro lines.) The **−ml** option forces the **−mm** option and also causes deletion of lists associated with the **mm** macros.

If the **−w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (&), and apostrophes ('); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

EXAMPLE

```
deroff textfile
```

removes all *nroff*, *troff*, and macro definitions from "textfile".

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1).

BUGS

Deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The **−ml** option does not handle nested lists correctly.

NAME

diff – differential file comparator

SYNOPSIS

diff [**–efbh**] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is –, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

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

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4* , are abbreviated as a single number.

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

The **–b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **–e** option produces a script of *a*, *c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **–f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

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

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **–h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **–e** and **–f** are unavailable with **–h**.

EXAMPLE

```
diff -e file1 file2
```

where "file1" and "file2" are two versions of the manual text for the *cp* command, produces:

```
35,41d
27c
In the second form, one or more


18,25c
existed; the mode of the source file
is used otherwise.

15c
The mode and owner of


10c
file ... directory


7c
file1 file2


1,3c
 .TH CP 1
 .SH NAME
```

Following this, *ed* script would transform "file1" into "file2", line for line and character for character.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

cmp(1), comm(1), ed(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the **−e** or **−f** option are naive about creating lines consisting of a single period (.).

If an unrecognized option is specified, *diff* ignores it and performs the default operation.

Diff may not work if files contain a very long line, or if files are very long.

WARNINGS

Missing newline at end of file X

indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

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
```

EXAMPLE

If file "f1" contains the following text:

```
This is a file.
This is the first of three files.
This is not the last file.
```

and file "f2" contains:

```
This is a file.
This is the second of three files.
This is not the last file.
```

and file "f3" contains:

```
This is a file.
This is the third of three files.
This is the last file.
```

then

```
diff3 f1 f2 f3
```

will return

```
====
1:2,3c
      This is the first of three files.
      This is not the last file.
```

```
2:2,3c
     This is the second of three files.
     This is not the last file.
3:2,3c
     This is the third of three files.
     This is the last file
```

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat −e.
Files longer than 64K bytes won't work.

NAME

diffdir – diff directories

SYNOPSIS

diffdir [**–h**] [**–s**] dir1 dir2

DESCRIPTION

Diffdir compares the differences of two directories recursively by sorting the contents of the directories by name and then runs a *diff* on text files which are different. Object files which differ and files which appear in only one directory are also listed.

The **–h** option causes *diffdir* to paginate its output, and to summarize binary differences and files in only one place at the end of the *diff*. Each individual *diff* is run through an appropriate *pr*.

The **–s** option causes files which are the same to be reported; normally they are omitted.

EXAMPLE

```
diffdir dir1 dir2
```

compares all the files in two directories and reports differences, by line number, for similar files. Unique files are simply listed.

FILES

/bin/cmp	compare two files

SEE ALSO

diff(1).

BUGS

Program should pass flags through to *diff*.

AUTHOR

Bill Joy

NAME

diffmk − mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff*(1) or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked.

EXAMPLE

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

produces a listing of two versions of a C program with changes marked. First the two versions are compared and a new file, "tmp", is created containing the *change mark* commands. The temporary file is then passed to *nroff*(1) using the file "macs" which contains:

```
.pl 1
.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.

SEE ALSO

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

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing .sp by .sp 2 will produce a "change mark" on the preceding or following line of output.

NAME

dircmp − directory comparison

SYNOPSIS

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

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same contents.

−d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

−s Suppress messages about identical files.

−w *n*
Change the width of the output line to *n* characters. The default width is 72.

EXAMPLE

```
dircmp d1 d2
```

will show the differences between the directories **d1** and **d2**.

SEE ALSO

cmp(1), diff(1).

DIRNAME(1) SEE *BASENAME* DIRNAME(1)

NAME

dis − disassembler

SYNOPSIS

dis [**−o**] [**−V**] [**−L**] [**−d** sec] [**−da** sec] [**−F** function] [**−t** sec] [**−l** string] files

DESCRIPTION

The *dis* command produces an assembly language listing of each of its object *file* arguments. The listing includes assembly statements and the binary that produced those statements.

The following options are interpreted by the disassembler and may be specified in any order.

−o Print numbers in octal. Default is hexadecimal.

−V Write the version number of the disassembler to standard error.

−L Invoke a lookup of C source labels in the symbol table for subsequent printing.

−d *sec* Disassemble the named section as data, printing the offset of the data from the beginning of the section.

−da *sec* Disassemble the named section as data, printing the actual address of the data.

−t *sec* Disassemble the named section as text.

−l *string* Disassemble the library file specified as *string*. For example, one would issue the command **dis −l x −l z** to disassemble **libx.a** and **libz.a**. All libraries are assumed to be in **/lib**.

If the **−d**, **−da**, or **−t** options are specified, only those named sections from each user supplied filename are disassembled. Otherwise, all sections containing text are disassembled.

If the **−F** option is specified, only those named functions from each user supplied filename are disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as [5], represents that the C breakpointable line number starts with the following instruction. An expression such as <40> in the operand field, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A C function name will appear in the first column, followed by ().

SEE ALSO

as(1), *cc*(1), *ld*(1).

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NAME

dis – disassembler

SYNOPSIS

dis [**–d** *sec*] [**–da** *sec*] [**–F** *function*] [**–l** *string*] [**–L**] [**–o**] [**–t** *sec*] [**–V**] *files*

DESCRIPTION

The **dis** command produces an assembly language listing of each of its object *file* arguments. The listing includes assembly statements and the binary that produced those statements.

The following flag options are interpreted by the disassembler and may be specified in any order:

–o Print numbers in octal. Default is hexadecimal.

–V Write the version number of the disassembler to standard error.

–L Invoke a lookup of C source labels in the symbol table for subsequent printing.

–d *sec* Disassemble the named section as data, printing the offset of the data from the beginning of the section.

–da *sec* Disassemble the named section as data, printing the actual address of the data.

–t *sec* Disassemble the named section as text.

–l *string* Disassemble the library file specified as *string*. For example, one would issue the command dis –l x –l z to disassemble libx.a and libz.a. All libraries are assumed to be in /lib.

If the **–d**, **–da**, or **–t** flag options are specified, only those named sections from each user-supplied filename are disassembled. Otherwise, all sections containing text are disassembled.

If the **–F** flag option is specified, only those named functions from each user-supplied filename are disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as [5], means that **dis** has reached the point in the assembly code where a C language line (numbered as stated) begins. If a breakpoint is placed there using sdb/adb, the debugger used will stop on a C line. An expression such as <40> in the operand field, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A C function name will appear in the first column, followed by ( ).

FILES

/bin/dis

SEE ALSO

as(1), cc(1), ld(1), strings(1).

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NOTES

The 68020 version of dis will disassemble 68881 and 68851 instructions, whether or not these are supported in the hardware. The 68000/68010 version will not disassemble 68881 or 68851 instructions or 68020 instructions not supported by the 68010.

SEE *ENABLE*

NAME

du – summarize disk usage

SYNOPSIS

du [**–ars**] [names]

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The default system size for physical blocks is 512 bytes. The block count includes the indirect blocks of the file. If *names* is missing, . is used.

The optional argument **–s** causes only the grand total (for each of the specified *names*) 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.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. The **–r** option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

EXAMPLE

```
du dir1 dir2
```

produces a count of the number of blocks in each of the directories. In order to see how many blocks are in each file, the **–a** option must be used.

BUGS

If the **–a** option is not used, non-directories given as arguments are not listed.
If there are too many distinct linked files, *du* will count the excess files more than once.
Files with holes in them will get an incorrect block count.

NAME

dump – dump selected parts of an object file

SYNOPSIS

dump [[**–a**] [**–c**] [**–f**] [**–g**] [**–h**] [**–l**] [**–o**] [**–r**] [**–s**] [**–t**] [**–z** *name*]] [[**–d** *number*] [**+d** *number*] [**-n** *name*] [**-p**] [**–t** *index*] [**+t** *index*] [**-u**] [**-v**] [**–z** *name,number*] [**+z** *name*]] *files*

DESCRIPTION

The **dump** command dumps selected parts of each of its object *file* arguments.

This command accepts both object files and archives of object files. It processes each file argument according to one or more of the following flag options:

–a Dump the archive header of each member of each archive file argument.

–f Dump each file header.

–g Dump the global symbols in the symbol table of a version 6.0 archive.

–o Dump each optional header.

–h Dump section headers.

–s Dump section contents.

–r Dump relocation information.

–l Dump line number information.

–t Dump symbol table entries.

–z *name* Dump line number entries for the named function.

–c Dump the string table.

The following modifiers are used in conjunction with the flag options listed above to modify their capabilities.

–d *number* Dump the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by **+d**.

+d *number* Dump sections in the range either beginning with first section or beginning with section specified by **–d**.

–n *name* Dump information pertaining only to the named entity. This *modifier* applies to **–h**, **–s**, **–r**, **–l**, and **–t**.

–p Suppress printing of the headers.

–t *index* Dump only the indexed symbol table entry. When the **–t** is used in conjunction with **+t**, it specifies a range of symbol table entries.

+t *index* Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **−t** flag option.

−u Underline the name of the file for emphasis.

−v Dump information in symbolic representation rather than numeric (e.g., **C_STATIC** instead of **0X02**). This *modifier* can be used with all the above flag options except the **−s** and **−o** flag options of **dump**.

−z *name,number*
Dump line number entry or range of line numbers starting at *number* for the named function.

+z *number* Dump line numbers starting at either function *name* or *number* specified by **−z**, up to *number* specified by **+z**.

Blanks separating a flag option and its modifier are optional. The comma separating the name from the number modifying the **−z** flag option may be replaced by a blank.

The **dump** command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal, or decimal representation, as appropriate.

FILES

/bin/dump

SEE ALSO

as(1), **dis**(1), **od**(1), **nm**(1), **strings**(1), **dumpfs**(1M), **restore**(1M), **a.out**(4), **ar**(4).

NAME

echo − echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\_n_	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

EXAMPLE

```
        echo curmudgeon
```

simply responds

```
        curmudgeon
```

on the standard output.

SEE ALSO

sh(1).

NAME

ed, red – text editor

SYNOPSIS

ed [–] [–pstring] [–x] [file]

red [–] [–pstring] [–x] [file]

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional – suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a !*shell command*. The **–p** option allows the user to specify a prompt string. If **–x** is present, an **X** command is simulated first to handle an encrypted file. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via !*shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec* (4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty –tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer.

While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

   a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([ ]; see 1.4 below).

   b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([ ]) (see 1.4 below).

   c. $ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).

   d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([ ]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (–) may be used to indicate a range of consecutive ASCII characters; for example, **[0–9]** is equivalent to **[0123456789]**. The – loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [ ]a–f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

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

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m,*\}, or \{*m,n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m,*\} matches *at least m* occurrences; \{*m,n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

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

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A currency symbol ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^*entire RE* $ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally 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 each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x* addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below. If *x* was not used to mark a line, '*x* addresses line 0.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or −, the addition or subtraction is taken with respect to the current line; e.g, −5 is understood to mean .−5.
9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so — refers to the current line less 2.

10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the current line (**.**) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n** or **p**, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(**.**)**a**
<text>
.
The *a*ppend command reads the given text and appends it after the addressed line; **.** is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(**.**)**c**
<text>
.
The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; **.** is left at the last line input, or, if there were none, at the first line that was not deleted.

(**.**,**.**)**d** The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file* The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file* The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

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

(1 , $)g/*RE*/*command list*

In the *g* lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1 , $)G/*RE*/

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

h The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
. The *i* nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x* The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address '*x* then addresses this line; . is unchanged.

(.,.)l The *l*ist 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 be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a* The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed.

The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

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

($)r *file* The *r* ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/g or
(.,.)s/*RE*/*replacement*/**n**

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number **n** appears after the command, only the **n**th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

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

u The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1 , $)v/*RE*/*command list*

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

(1 , $)V/*RE*/

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

(1 , $)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)= The line number of the addressed line is typed; address 0 is legal for this command. . is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the UNIX system shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (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.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

EXAMPLE

ed text

would invoke the editor with the file named "text". For further examples, see **ED** in the *User Guide*.

FILES

/tmp/e# temporary; # is the process number.

ed.hup work is saved here if the terminal is hung up.

DIAGNOSTICS

? for command errors.

?file for an inaccessible file.

(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands. It prints ? and allows one to continue editing. A second *e* or *q* command will take effect at this point or at any later time, provided no further changes have been made to the file. The – command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1), stty(1), fspec(4), regexp(5).
ED in the *User Guide*.

CAVEATS AND BUGS

The *!* command and the **!** escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).
The sequence **\n** in a RE does not match a new-line character.
The *l* command mishandles DEL.
Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.
Characters are masked to 7 bits on input.

NOTE

The **–x** option and the editor command **X** are not implemented in the international distribution.
If the editor input is coming from a command file (i.e., ed file < ed-cmd-file), the editor will exit at the first failure of a command that is in the command file.

SEE *EX*

NAME

efl − Extended Fortran Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the EFL language into clean Fortran on the standard output. *Efl* provides the C-like control constructs similar to *ratfor*:

statement grouping with braces.

decision-making:
if, if-else, and **select-case** (also known as **switch-case**);
while, for, Fortran **do, repeat**, and **repeat** ... **until** loops;
multi-level **break** and **next**.

EFL has C-like data structures, e.g.:

```
struct
        {
        integer flags(3)
        character(8) name
        long real coords(2)
        } table(100)
```

The language offers generic functions, assignment operators (+=, &=, etc.), and sequentially evaluated logical operators (**&&** and **||**). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:
this is a comment.

translation of relational and logical operators:
>, **>=**, **&**, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:
return (*expression*)

defines:
define *name replacement*

includes:
include *file*

Efl understands several option arguments: **−w** suppresses warning messages, **−#** suppresses comments in the generated program, and the default option **−C** causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on. Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

Efl is best used with *f77*(1).

EXAMPLE

```
efl prog.for | f77 -o prog
```

will process the program *prog.for* through *efl* and then run the *f77*(1) compiler on the output from *efl*, generating an executable file named "prog".

SEE ALSO

cc(1), f77(1), ratfor(1).
EFL in the *Programming Guide*.

NAME

enable, disable − enable/disable LP printers

SYNOPSIS

enable printers
disable [**−c**] [**−r**[reason]] printers

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

−c Cancel any requests that are currently printing on any of the designated printers.

−r[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **−r** option. If the **−r** option is not present or the **−r** option is given without a reason, then a default reason will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env − set environment for command execution

SYNOPSIS

env [−] [name=value] ... [command args]

DESCRIPTION

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The − flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

EXAMPLE

```
env XYZ=pdq sh
```

sets the environment name "XYZ" to the value *pdq* for the duration of the new shell.

SEE ALSO

sh(1), exec(2), profile(4), environ(5).

NAME

eqn, neqn, checkeq − format mathematical text for nroff or troff

SYNOPSIS

eqn [**−d**xy] [**−p**n] [**−s**n] [**−f**n] [files]

neqn [**−d**xy] [**−p**n] [**−s**n] [**−f**n] [files]

checkeq [files]

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematical text on a phototypesetter, while *neqn* is used for the same purpose with *nroff* on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent.

If no files are specified (or if − is specified as the last argument), these programs read the standard input. A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a line beginning with .EN. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **−d***xy* or (more commonly) with **delim** *xy* between .EQ and .EN. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub j* makes x_j, *a sub k sup 2* produces a_k^2, while $e^{x^2+y^2}$ is made with *e sup {x sup 2 + y sup 2}*. Fractions are made with **over**: *a over b* yields $\frac{a}{b}$; **sqrt** makes square roots: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits: $\lim\limits_{n\to\infty}\sum\limits_0^n x_i$ 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 $\left[x^2+\frac{y^2}{\alpha}\right] = 1$.

Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix}a\\ b.\\ c\end{matrix}$ Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies. Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix}x_i & 1\\ y_2 & 2\end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x}=\overline{f(t)}$, *y dotdot bar* ¯ =¯ *n under* is $\bar{y} = \underline{n}$, and *x vec* ˜ =˜ *y dyad* is $\bar{x} = \bar{y}$.

Point sizes and fonts can be changed with **size** *n* or **size** ± *n*, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s***n* and **−f***n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **−p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

```
define thing % replacement %
```

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** ($\sum$), **int** ($\int$), **inf** (∞), and shorthands such as >= ($\geqslant$), != ($\neq$), and −> ($\rightarrow$) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or GAMMA (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as **\(dd** (‡) and **\(bs** (🔔) may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff*(1) when all else fails. Full details are given in the manual cited below.

EXAMPLE

```
eqn file1 | troff
```

would process the file "file1" with the preprocessor before formatting it with *troff.*

SEE ALSO

cw(1), mm(1), mmt(1), nroff(1), tbl(1), troff(1), eqnchar(5), mm(5), mv(5).
EQN in the *Document Processing Guide*.

BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.
See also *BUGS* under *troff*(1).

NAME

ex, edit – text editor

SYNOPSIS

ex [–] [–v] [–ttag] [–r] [–R] [+command] [–x] name ...
edit [ex options]

DESCRIPTION

Ex is the root of a family of editors: *edit*, *ex* and *vi*. *Ex* is a superset of *edit*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have not used *ed*, or are a casual user, you will find that the editor *edit* is convenient for you. It avoids some of the complexities of *ex* used mostly by systems programmers and persons very familiar with *ed*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

The following options are recognized:

– Suppresses all interactive-user feedback, as when processing editor scripts in command files. 348.sp 38u

–v Equivalent to using *vi* rather than *ex*.

–t *tag*
Equivalent to an initial *tag* command, editing the file containing the *tag* and positioning the editor at its definition.

–r *file*
Used in recovering after an editor or system crash, retrieving the last saved version of the named file. If no file is specified, a list of saved files will be reported.

–R *Readonly* mode set, prevents accidentally overwriting the file.

+*command*
Indicates that the editor should begin by executing the specified command. If *command* is omitted, then it defaults to $, positioning the editor at the last line of the first file initially. Other useful commands here are scanning patterns of the form */pat* or line numbers, e.g., +100 to start at line 100.

–x Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

name

Indicates files to be edited.

Ex States

Command — Normal and initial state. Input prompted for by :. Your kill character cancels partial command.

Insert — Entered by **a i** and **c.** Arbitrary text may be entered. Insert is normally terminated by line having only . on it, or abnormally with an interrupt.

Visual — Entered by **vi**, terminates with **Q** or ^\

Ex command names and abbreviations

abbrev	**ab**	next	**n**	undo	**u**
append	**a**	number	**nu**	unmap	**unm**
args	**ar**	preserve	**pre**	version	**ve**
change	**c**	print	**p**	visual	**vi**
copy	**co**	put	**pu**	write	**w**
delete	**d**	quit	**q**	xit	**x**
edit	**e**	read	**re**	yank	**ya**
file	**f**	recover	**rec**	window	**z**
global	**g**	rewind	**rew**	escape	**!**
insert	**i**	set	**se**	lshift	**<**
join	**j**	shell	**sh**	printnext	**CR**
list	**l**	source	**so**	resubst	**&**
map	**map**	stop	**st**	rshift	**>**
mark	**ma**	substitute	**s**	scroll	**^D**
move	**m**	unabbrev	**una**		

Ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
–	previous	*'x*	marked with *x*
+n	*n* forward	''	previous context
%	1,$		

Initializing options

EXINIT	place set's here in environment var.
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file

set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp	lisp	( ) { } are s-exp's
list	list	print ^I for tab, $ at end
magic	magic	. [* special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw	redraw	simulate smart terminal
scroll	scroll	command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in *vi*
slowopen	slow	stop updates during insert
window	window	visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any char in *str*
[↑*str*]	... not in *str*
[*x*–*y*]	... between *x* and *y*
*	any number of preceding

AUTHOR

Vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical

Engineering and Computer Science.

Documentation

The *User Guide* provides editing tutorials for *ex*, *ed*, and *vi*.

FILES

/usr/lib/ex3.9strings	error messages
/usr/lib/ex3.9recover	recover command
/usr/lib/ex3.9preserve	preserve command
/usr/lib/*/*	describes capabilities of terminals
~/.exrc	editor startup command file, user-created in home directory
/tmp/EX*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory
/usr/bin/ctags	standard editor tag file

EXAMPLE

```
ex text
```

would invoke the editor with the file named "text".

SEE ALSO

awk(1), ed(1), grep(1), sed(1), vi(1) curses(3X), term(4), terminfo(4).

BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line "–" option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

NAME

expr − evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

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

expr \| *expr*
: returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr*
: returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
: returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, − } *expr*
: addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
: multiplication, division, or remainder of the integer-valued arguments.

expr : *expr*
: The matching operator : compares the first argument with the second argument which must be a regular expression; regular expression syntax is the same as that of *ed*(1), except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

EXAMPLE

```
a=`expr $a + 1`
```

adds 1 to the shell variable **a**.

```
# 'For $a equal to either "/usr/abc/file" or just "file"'
expr $a : '.*/\(.*\)' \| $a
```

returns the last segment of a path name (i.e., "file"). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

```
# A better representation of above example
expr //$a : '.*/\(.*\)'
```

the addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

```
expr $VAR : '.*'
```

returns the number of characters in **$VAR**.

SEE ALSO

ed(1), sh(1).

EXIT CODE

As a side effect of expression evaluation, *expr* returns the following exit values:

0 if the expression is neither null nor 0
1 if the expression *is* null or 0
2 for invalid expressions.

DIAGNOSTICS

syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

NAME

exterr - turn on/off the extended errors in the specified device

SYNOPSIS

exterr /dev/devicename [yn]

DESCRIPTION

Exterr turns on (or off) the reporting of extended errors on the specified device.

If reporting of errors is turned "off" with the argument **n**, only fatal errors are reported.

The default condition is "yes", in which case soft as well as hard errors are reported on the specified device. The devicename must be the "raw" one to access the *ioctl*.

EXAMPLE

```
exterr /dev/xxxx n
```

turns to off the reporting of extended errors for device **/dev/xxxx**.

NAME

f77 – Fortran 77 compiler

SYNOPSIS

f77 [options] files

DESCRIPTION

F77 is the Fortran 77 compiler; it accepts several types of *file* arguments:

> Arguments whose names end with **.f** are taken to be Fortran 77 source programs; they are compiled and each object program is left in the current directory in a file whose name is that of the source, with **.o** substituted for **.f**.
>
> Arguments whose names end with **.r** or **.e** are taken to be RATFOR or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by *f77*, producing **.o** files.
>
> In the same way, arguments whose names end with **.c** or **.s** are taken to be C or assembly source programs and are compiled or assembled, producing **.o** files.

The following options have the same meaning as in *cc*(1) (see *ld*(1) for link editor options):

–c Suppress link editing and produce **.o** files for each source file.

–p Prepare object files for profiling (see *prof*(1)).

–O Invoke an object code optimizer.

–S Compile the named programs and leave the assembler language output in corresponding files whose names are suffixed with **.s**. (No **.o** files are created.)

–o*output* Name the final output file *output*, instead of **a.out**.

–f In systems without floating-point hardware, use a version of *f77* that handles floating-point constants and links the object program with the floating-point interpreter.

–g Generate additional information needed for the use of *sdb*(1)

The following options are peculiar to *f77*:

–onetrip Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

–1 Same as **–onetrip**.

−66 Suppress extensions which enhance Fortran 66 compatibility.

−C Generate code for run-time subscript range-checking.

−I[24s] Change the default size of integer variables (only valid on machines where the ''normal'' integer size is not equal to the size of a single precision real). **−I2** causes all integers to be 2-byte quantities, **−I4** (default) causes all integers to be 4-byte quantities, and **−Is** changes the default size of subscript expressions (only) from the size of an integer to 2 bytes.

−U Do not "fold" cases. *F77* is normally a no-case language (i.e. a is equal to **A**). The **−U** option causes *f77* to treat upper and lower cases separately.

−u Make the default type of a variable *undefined*, rather than using the default Fortran rules.

−w Suppress all warning messages. If the option is **−w66**, only Fortran 66 compatibility warnings are suppressed.

−F Apply EFL and RATFOR preprocessor to relevant files and put the result in files whose names have their suffix changed to **.of**. (No **.o** files are created.)

−m Apply the M4 preprocessor to each EFL or RATFOR source file before transforming with the *ratfor*(1) or *efl*(1) processors.

−E The remaining characters in the argument are used as an EFL flag argument whenever processing a **.e** file.

−R The remaining characters in the argument are used as a RATFOR flag argument whenever processing a **.r** file.

Other arguments are taken to be link editor option arguments, *f77*-compilable object programs (typically produced by an earlier run), or libraries of *f77*-compilable routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the default name **a.out**.

FILES

file.[fresc]	input file
file.o	object file
a.out	linked output
./fort[*pid*].?	temporary
/usr/lib/f77comp	compiler
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library

/lib/libc.a	C library; see Section 3 of this Manual.

SEE ALSO

FORTRAN-77 in the *Programming Guide*.
asa(1), cc(1), efl(1), fsplit(1), ld(1), m4(1), prof(1), ratfor(1), sdb(1).

DIAGNOSTICS

The diagnostics produced by *f77* itself are self-explanatory. Occasional messages may be produced by the link editor *ld* (1).

NAME

factor — factor a number

SYNOPSIS

factor [number]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about 7.2×10^{16}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to $\sqrt{n}$ and occurs when n is prime or the square of a prime. It takes 30 seconds to factor a prime near 10^{14} on a 68000.

DIAGNOSTICS

"Ouch" for input out of range or for garbage input.

FALSE(1) **SEE** ***TRUE*** **FALSE(1)**

FGREP(1) **SEE** ***GREP*** **FGREP(1)**

NAME

file − determine file type

SYNOPSIS

file [**−c**] [**−f** ffile] [**−m** mfile] arg ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0 (see *ld*(1)).

If the **−f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

File uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

The **−m** option instructs *file* to use an alternate magic file.

The **−c** flag causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **−c**.

EXAMPLE

```
file textfile programfile directory
```

reports the file names and directory name, and whether the files are English text, *nroff* input, a C program, or whatever.

SEE ALSO

ld(1).

NAME

find – find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) 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 *file* True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for [, ? and *).

–perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

(flags & *onum*) == *onum*

–type *c*

True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file respectively.

–links *n*

True if the file has *n* links.

–user *uname*

True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

–group *gname*

True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

–size *n*[**c**]

True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a c, the size is in characters.

–atime *n*

True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.

−mtime *n*
True if the file has been modified in *n* days.

−ctime *n*
True if the file has been changed in *n* days.

−exec *cmd*
True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.

−ok *cmd*
Like −exec except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

−print
Always true; causes the current path name to be printed.

−cpio *device*
Always true; write the current file on *device* in *cpio*(4) format (5120-byte records).

−newer *file*
True if the current file has been modified more recently than the argument *file*.

−depth
Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.

(*expression*)
True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (! is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**−o** is the *or* operator).

EXAMPLE

```
find / -perm 755 -exec ls "{}" ";"
```

will find all files, starting with the root directory, on which the permission levels have been set to 755 (see *chmod* (1)).

With –exec and a command such as *ls*, it is often necessary to escape the "{}" that stores the current pathname under investigation by putting it in double quotes. It is always necessary to escape the semicolon at the end of an –exec sequence.

Note again that it is also necessary to escape parentheses " \(" and " \) " used for grouping primaries, by means of a backslash.

```
find / \( –name a.out –o –name '*.o' \) –atime +7 –exec rm {} \;
```

removes all files named "a.out" or "*.o" that have not been accessed for a week.

FILES

/etc/passwd, /etc/group

SEE ALSO

chmod(1), cpio(1), sh(1), test(1), stat(2), cpio(4), fs(4).

NAME

freq − report on character frequencies in a file

SYNOPSIS

freq [file ...]

DESCRIPTION

Freq counts occurrences of characters in the list of files specified on the command line. If no files are specified, the standard input is read.

EXAMPLE

```
freq filea
```

will list a count of each character that appears in "filea".

NAME

fsplit − split f77, ratfor, or efl files

SYNOPSIS

fsplit options files

DESCRIPTION

Fsplit splits the named *file* (*s*) into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Procedure *X* is put in file *X* .f, *X* .r, or *X*.e depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN*.|efr| and unnamed *blockdata* segments in the files *blockdataN*.|efr| where *N* is a unique integer value for each file.

The following *options* pertain:

−f (default) Input files are *f77*.

−r Input files are *ratfor*.

−e Input files are *Efl*.

−s Strip *f77* input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), efl(1), f77(1), ratfor(1), split(1).

NAME

ftp – file transfer program

SYNOPSIS

ftp [–v] [–d] [–i] [–n] [–g] [host]

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user the prompt ftp> is provided the user. The following commands are recognized by *ftp*:

! Invoke a shell on the local machine.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

ascii
Set the file transfer *type* to network ASCII. This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary
Set the file transfer *type* to support binary image transfer.

bye Terminate the FTP session with the remote server and exit *ftp*.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close
Terminate the FTP session with the remote server, and return to the command interpreter.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used

to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string -->.

dir [*remote-directory*] [*local-file*]

Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal.

form *format*

Set the file transfer *form* to *format*. The default format is file.

get *remote-file* [*local-file*]

Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

hash

Toggle hash-sign (''#'') printing for each data block transferred. The size of a data block is 1024 bytes.

glob

Toggle file name globbing. With file name globbing enabled, each local file or pathname is processed for *csh*(1) metacharacters. These characters include ''*?[]~{}''. Remote files specified in multiple item commands, e.g. *mput*, are globbed by the remote server. With globbing disabled all files and pathnames are treated literally.

help [*command*]

Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]

Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]

Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

mdelete *remote-files*

Delete the specified files on the remote machine. If globbing is enabled, the specification of remote files will first be expanded using *ls*.

mdir *remote-files local-file*
Obtain a directory listing of multiple files on the remote machine and place the result in *local-file*.

mget *remote-files*
Retrieve the specified files from the remote machine and place them in the current local directory. If globbing is enabled, the specification of remote files will first be expanding using *ls*.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Obtain an abbreviated listing of multiple files on the remote machine and place the result in *local-file*.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is stream mode.

mput *local-files*
Transfer multiple local files from the current local directory to the current working directory on the remote machine.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default), any *mget* or *mput* will transfer all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd
Print the name of the current working directory on the remote machine.

quit
A synonym for bye.

quote *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

recv *remote-file* [*local-file*]
A synonym for get.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for put.

sendport
Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status
Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default stream structure is used.

tenex
Set the file transfer type to that needed to talk to TENEX machines.

trace
Toggle packet tracing.

type [*type-name*]

Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [*password*] [*account*]

Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. Unless *ftp* is invoked with auto-login disabled, this process is done automatically on initial connection to the FTP server.

verbose

Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]

A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name – is specified, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is |, the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell, using *popen*(3S) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. "| ls -lt". A particularly useful example of this mechanism is: "dir . |more".

3) Failing the above checks, if ''globbing'' is enabled, local file names are expanded according to the rules used in the *csh*(1); c.f. the *glob* command.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of ascii, image (binary), ebcdic, and local byte size (for PDP-10's and PDP-20's mostly). *Ftp* supports the ascii and image types of file transfer.

Ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

The **−v** option toggles verbose mode. In verbose mode, all responses from the remote *ftp* server are displayed, as well as statistics on data transfer efficiency. By default, verbose is on.

The **−n** option restrains *ftp* from attempting auto-login upon initial connection. If auto-login is enabled, *ftp* will check the *.netrc* file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will use the login name on the local machine as the user identity on the remote machine, and prompt for a password and, optionally, an account with which to login.

The **−i** option turns off interactive prompting during mutliple file transfers.

The **−d** option enables debugging.

The **−g** option disables file name globbing.

BUGS

Correct execution of many depends upon proper behavior by the remote server.

Many FTP server implementations do not support some operations such as print working directory. When verbose mode (−v) is turned off, *ftp* does not echo responses from the remote server. This includes the response to the *pwd* request. Beware of this.

Aborting a file transfer does not work right; if one attempts this the local *ftp* will likely have to be killed by hand.

NAME

get − get a version of an SCCS file

SYNOPSIS

get [**−r**SID] [**−c**cutoff] [**−i**list] [**−x**list] [**−w**string] [**−a**seq-no.] [**−k**] [**−e**] [**−l**[**p**]] [**−p**] [**−m**] [**−n**] [**−s**] [**−b**] [**−g**] [**−t**] file ...

DESCRIPTION

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

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

−r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the **−e** keyletter is also used), as a function of the SID specified.

−c*cutoff* *Cutoff* date-time, in the form: YY[MM[DD[HH[MM[SS]]]]] No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "**−c77/2/2 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*(2N) command:

```
~!get "−c%E% %U%" s.file
```

−e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The **−e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get −e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **−e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **−k** keyletter in place of the **−e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **−e** keyletter is used.

−b Used with the **−e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

−i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID − SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

−x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **−i** keyletter for the *list* format.

−k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **−k** keyletter is implied by the **−e** keyletter.

−l[p] Causes a delta summary to be written into an *l-file*. If **−lp** is used, then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

−p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **−s** keyletter is used, in which case it disappears.

−s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

−m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line.

When both the **−m** and **−n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** keyletter generated format.

−g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Used to access the most recently created ("top") delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).

−w*string* Substitute *string* for all occurrences of **%W%** when *get*ing the file.

−a*seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the **−r** and **−a** keyletters are specified, the **−a** keyletter is used. Care should be taken when using the **−a** keyletter in conjunction with the **−e** keyletter, as the SID of the delta to be created may not be what one expects. The **−r** keyletter can be used with the **−a** and **−e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** keyletter is used included deltas are listed following the notation "Included"; if the **−x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**−b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	−	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	−	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	−	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1

R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

# Successor.

† The **−b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of − means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword *Value*

%M% Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.

%I% SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.

%R% Release.

%L% Level.

%B% Branch.

%S% Sequence.

%D% Current date (YY/MM/DD).

%H% Current date (MM/DD/YY).

%T% Current time (HH:MM:SS).

%E% Date newest applied delta was created (YY/MM/DD).

%G% Date newest applied delta was created (MM/DD/YY).

%U% Time newest applied delta was created (HH:MM:SS).

%Y% Module type: value of the **t** flag in the SCCS file (see *admin*(1)).

%F% SCCS file name.

%P% Fully qualified SCCS file name.

%Q% The value of the **q** flag in the file (see *admin*(1)).

%C% Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.

%Z% The 4-character string **@(#)** recognizable by *what*(1).

%W% A shorthand notation for constructing *what*(1) strings for UNIX system program files. %W% = %Z%%M%<horizontal-tab>%I%

%A% Another shorthand notation for constructing *what*(1) strings for non-UNIX system program files.
%A% = %Z%%Y% %M% %I%%Z%

EXAMPLE

```
get -e s.file1
```

generates from the SCCS format file, "s.file1", the text file, "file1", for editing.

FILES

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

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

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

i. Login name of person who created *delta*.

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

The *p-file* is used to pass information resulting from a *get* with an **−e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **−e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **−i** keyletter argument if it was present, followed by a blank and the **−x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

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

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(4).
SCCS in the *Programming Tools Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the **−e** keyletter is used.

NAME

getopt − parse command options

SYNOPSIS

set −− \`getopt optstring $\*\`

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters (see *getopt*(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option −− is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The shell's positional parameters ($1 $2 ...) are reset so that each option is preceded by a − and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set −− `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        −a | −b)        FLAG=$i; shift;;
        −o)             OARG=$2; shift 2;;
        −−)             shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd −aoarg file file
cmd −a −o arg file file
cmd −oarg −a file file
cmd −a −oarg −− file file
```

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

graph − draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *tplot*(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ", in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

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

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).

−b Break (disconnect) the graph after each label in the input.

−c Character string given by next argument is default label for each point.

−g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

−l Next argument is label for graph.

−m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

−s Save screen, don't erase before plotting.

−x [l] If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

−y [l] Similarly for *y*.

−h Next argument is fraction of space for height.

−w Similarly for width.

−r Next argument is fraction of space to move right before plotting.

−u Similarly to move up before plotting.

−t Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

tplot(1G).

BUGS

Graph stores all points internally and drops those for which there isn't room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

NAME

grep, egrep, fgrep − search a file for a pattern

SYNOPSIS

grep [options] expression [files]

egrep [options] [expression] [files]

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

−v All lines but those matching are printed.
−x (Exact) only lines matched in their entirety are printed (*fgrep* only).
−c Only a count of matching lines is printed.
−i Ignore upper/lowercase distinction during comparisons.
−l Only the names of files with matching lines are listed (once), separated by new-lines.
−n Each line is preceded by its relative line number in the file.
−b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
−s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).
−e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a − (does not work with *grep*).
−f *file*
The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes **'**...**'**.

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

Egrep accepts regular expressions as in *ed*(1), except for **\\(** and **\\)**, with the addition of:

1. A regular expression followed by **+** matches one or more occurrences of the regular expression.
2. A regular expression followed by **?** matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by **|** or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses **()** for grouping.

The order of precedence of operators is **[]**, then *** ? +**, then concatenation, then **|** and new-line.

EXAMPLE

```
grep -v -c 'regular' grep.1
```

reports a count of the number of lines that do not contain the word *regular* in the file "grep.1".

SEE ALSO

ed(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we do not know a single algorithm that spans a wide enough range of space-time tradeoffs.
Lines are limited to BUFSIZ characters; longer lines are truncated. (BUFSIZ is defined in **/usr/include/stdio.h.**)
Egrep does not recognize ranges, such as **[a−z]**, in character classes.
If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

greek − select terminal filter

SYNOPSIS

greek [−Tterminal]

DESCRIPTION

Greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE® Teletypewriter Model 37 terminal (which is the *nroff* default terminal) for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, *greek* attempts to use the environment variable **$TERM** (see *environ*(5)). The following *terminal*s are recognized currently:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

EXAMPLE

```
nroff filename | greek -T4014
```

reinterprets the extended character set on a Tektronix 4014 terminal.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), eqn(1), mm(1), nroff(1), tplot(1G), environ(5), greek(5), term(5).

HASHCHECK(1) **SEE *SPELL*** **HASHCHECK(1)**

HASHMAKE(1) **SEE *SPELL*** **HASHMAKE(1)**

NAME

hex − translates object files

SYNOPSIS

hex [**−f**] [**−l**] [**−n#**] [**−r**] [**−s0**] [**−s2**] [**−ns8**] [**+saddr**] ifile

DESCRIPTION

Hex translates object files into ASCII formats suitable for Motorola S-record downloading. The following options determine locations:

f The file specified is to be shipped as is without treating it as an object file.

l Output "Loading at" message.

n# Number of characters to output per record. # is a decimal number.

r Output a carriage return at the end of each S-record (instead of a newline).

s0 Output a leading **s0** record.

s2 S2 records only (no **s1** records are produced).

ns8 Do not output a trailing **s8** (s9) record.

saddr Starting load address (in hex).

ifile File to be downloaded. The file's starting address is used if **saddr** is not present.

EXAMPLE

```
hex objfile
```

where "objfile" is the object file to be downloaded.

AUTHOR

Jeff Schriebman

NAME

help — ask for help in using SCCS

SYNOPSIS

help [args]

DESCRIPTION

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

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

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **ge4**, 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., **26**)

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".

EXAMPLE

```
help he2
```

prints the message for error number "he2".

FILES

/usr/lib/help	directory containing files of message text.
/usr/lib/help/helploc	file containing locations of help files not in **/usr/lib/help**. This file is created by the user for user-defined help messages.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

head − give first few lines

SYNOPSIS

head [−count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

EXAMPLE

```
head -6 filea fileb filec
```

will print out the first six lines of the three specified files. The filename will appear before each new set of head lines listed, if more than one file has been specified.

SEE ALSO

tail(1).

NAME

hostid — set or print identifier of current host system

SYNOPSIS

hostid [identifier]

DESCRIPTION

The *hostid* command prints the identifier of the current host in hexadecimal. This numeric value is expected to be unique across all hosts and is normally set to the host's Internet address. The super-user can set the hostid by giving a hexadecimal argument; this is usually done in the startup script /etc/sysinitrc.

SEE ALSO

gethostid(2N)

NAME

hostname − set or print name of current host system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the "login" prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script /etc/sysinitrc.

SEE ALSO

gethostname(2N)

NAME

hyphen — find hyphenated words

SYNOPSIS

hyphen [files]

DESCRIPTION

Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

EXAMPLE

If the file "text.hyphen" contains the following text:

```
This is an ex-
ample of the command hy-
phen, a com-
mand which finds all hyphen-
ated words in files and prints them on stan-
dard out-
put.
```

then

```
hyphen text.hyphen
```

will return

```
ex-ample
hy-phen
com-mand
hyphen-ated
stan-dard
out-put
```

SEE ALSO

mm(1), nroff(1).

BUGS

Hyphen cannot cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

id – print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

EXAMPLE

```
id
```

will return

```
uid=100 (guest) gid=100 (users)
```

where "100" and "guest" are the user's ID number and name and "100" and "users" are the user's group ID number and group name. These values are set up in the administrative file **/etc/passwd.**

SEE ALSO

logname(1), getuid(2).

NAME

ipcrm − remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm will remove one or more specified message, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

−q *msqid* removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

−m *shmid* removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

−s *semid* removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

−Q *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

−M *shmkey* removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

−S *semkey* removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

NAME

ipcs − report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

Ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

−q Print information about active message queues.
−m Print information about active shared memory segments.
−s Print information about active semaphores.

If any of the options **−q**, **−m**, or **−s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

−b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

−c Print creator's login name and group name. See below.

−o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

−p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

−t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop* (2) on semaphores.) See below.

−a Use all print *options*. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)

−C *corefile*
Use the file *corefile* in place of **/dev/kmem**.

−N *namelist*
The argument will be taken as the name of an alternate *namelist* (**/unix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:
q message queue;
m shared memory segment;
s semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags. The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

- **R** if a process is waiting on a *msgrcv*;
- **S** if a process is waiting on a *msgsnd*;
- **D** if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
- **C** if the associated shared memory segment is to be cleared when the first attach is executed;
- **–** if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the usergroup of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

- **r** if read permission is granted;
- **w** if write permission is granted;
- **a** if alter permission is granted;
- **–** if the indicated permission is *not* granted.

OWNER (all) The login name of the owner of the facility entry.

GROUP (all) The group name of the group of the owner of the facility entry.

CREATOR (a,c) The login name of the creator of the facility entry.

CGROUP (a,c) The group name of the group of the creator of the facility entry.

CBYTES (a,o) The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o) The number of messages currently outstanding on the associated message queue.

QBYTES (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p) The process ID of the last process to send a message to the associated queue.

LRPID (a,p) The process ID of the last process to receive a message from the associated queue.

STIME (a,t) The time the last message was sent to the associated queue.

RTIME (a,t) The time the last message was received from the associated queue.

CTIME (a,t) The time when the associated entry was created or changed.

NATTCH (a,o) The number of processes attached to the associated shared memory segment.

SEGSZ (a,b) The size of the associated shared memory segment.

CPID (a,p) The process ID of the creator of the shared memory entry.

LPID (a,p) The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,t) The time the last attach was completed to the associated shared memory segment.

DTIME (a,t) The time the last detach was completed on the associated shared memory segment.

NSEMS (a,b) The number of semaphores in the set associated with the semaphore entry.

OTIME (a,t) The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2).

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

join – relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is –, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. Thus, to preserve tabs and multiple occurrences of spaces in a file, you must select tabs as the alternate delimiter using the **–t** option where *c* is the tab character (see the **–t** option below). The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

–a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

–e *s* Replace empty output fields by string *s*.

–j*n m* Join on the *m* th field of file *n*. If *n* is missing, use the *m* th field in each file. Fields are numbered starting with 1.

–o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n*. *m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.

–t *c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

```
If "file1" contains:   Austen -
                       Bailey -
```

```
Clark -
Dawson -
Smith -
```

and "file2" contains:

```
Austen Jack Anchor Brewery
Clark Maryann Shoeshop
Daniels Steve Computer Software
Dawson Sylvia Toot Sweets
Smith Sally Talcum Powdery
```

then

```
join –j1 1 –j2 1 –o 2.2 2.1 1.2 2.3 2.4 file1 file2
```

will generate

```
Jack Austen - Anchor Brewery
Maryann Clark - Shoeshop
Sylvia Dawson - Toot Sweets
Sally Smith - Talcum Powdery
```

```
join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group
```

joins the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name, and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

SEE ALSO

awk(1), comm(1), sort(1) uniq(1).

BUGS

With default field separation, the collating sequence is that of *sort* **–b**; with **–t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk* (1) are wildly incongruous.

Filenames that are numeric may cause conflict when the -o option is used right before listing filenames.

NAME

kill − terminate a process

SYNOPSIS

kill [−signo] PID ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the Shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(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.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by − is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill −9 ..." is a sure kill.

EXAMPLE

```
kill 24068
```

Sends signal 15 to the process with the ID number 24068.

SEE ALSO

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

NAME

last − indicate last logins of users and teletypes

SYNOPSIS

last [name ...] [tty ...]

DESCRIPTION

Last will look back in the *wtmp* file which records all logins and logouts for information about a user, a teletype [terminal] or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example **last 0** is the same as **last tty0'** . If multiple arguments are given, the information which applies to any of the arguments is printed. For example **last root console** would list all of "root's" sessions as well as all sessions on the console terminal.

Last reports the sessions of the specified users and teletypes, most recent first, indicating start times, duration, and teletype for each. If the session is still continuing or was cut short by a reboot, *last* so indicates.

EXAMPLE

```
last reboot
```

will give an indication of mean time between reboots of the system.

Last with no arguments prints a record of all logins and logouts, in reverse order. Since *last* can generate a great deal of output, piping it through the *more* program for screen viewing is advised.

If *last* is interrupted with a "break", it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\), *last* exits and dumps core.

Control-d (EOF) signal does nothing. Therefore exit gracefully from *last* with a "break" or "shift/delete" signal.

FILES

/etc/wtmp	login data base

AUTHOR

Howard Katseff

NAME

lav − print load average statistics

SYNOPSIS

lav

DESCRIPTION

Lav prints the average number of jobs in the run queue over the last 1, 5, and 15 minutes.

NAME

ld – link editor for common object files

SYNOPSIS

ld [**–e***epsym*] [**–f***fill*] [**–l***x*] [**–m**] [**–o***outfile*] [**–r**] [**–s**] [**–t**] [**–u***symname*] [**–x**] [**–z**] [**–F**] [**–L***dir*] [**–M**] [**–N**] [**–V**] [**–VS***num*] *filenames*

DESCRIPTION

The **ld** command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and **ld** combines them, producing an object module that can either be executed or used as input for a subsequent **ld** run. The output of **ld** is left in **a.out**. This file is executable if no errors occurred during the load. If any input file, *filename*, is not an object file, **ld** assumes it is either a text file containing link editor directives or an archive library.

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. The library (archive) symbol table (see **ar**(4)) is searched sequentially with as many passes as are necessary to resolve external references that can be satisfied by library members. Thus, the ordering of library members is unimportant.

The following flag options are recognized by **ld**:

–e*epsym*
Set the default entry point address for the output file to be that of the symbol *epsym*.

–f*fill*
Set the default fill pattern for holes within an output section as well as initialized bss sections. The argument *fill* is a two-byte constant.

–l*x* Search a library **lib***x***.a**, where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a **–l** is significant. The default library location is **/lib**.

–m Produce a map or listing of the input/output sections on the standard output.

–o*outfile*
Produce an output object file by the name *outfile*. The name of the default object file is **a.out**.

–r Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent **ld** run. The link editor does not complain about unresolved references.

–s Strip line number entries and symbol table information from the output object file.

–t Turn off the warning about multiply-defined symbols that are not the same size.

–u*symname*
Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

–x Do not preserve local (non-global) symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

–z Load the text segment at an offset from zero so that null pointer references generate a segmentation violation.

–F Create a demand-paged executable.

–L*dir*
Change the algorithm of searching for libx.a to look in *dir* before looking in /lib and /usr/lib. This flag option is effective only if it precedes the –l flag option on the command line.

–M Output a message for each multiply-defined external definition. However, if the objects being loaded included debugging information, extraneous output is produced (see the **–g** option in cc(1).

–N Put the data section immediately following the text in the output file. Note that the –N option must be used either with /usr/lib/unshared.ld or with a user-supplied *.ld* file.

–V Output a message giving information about the version of ld being used.

–VS*num*
Use *num* as a decimal version stamp identifying the a.out file that is produced. The version stamp is stored in the optional header.

The following information about section alignment and MMU requirements should be considered at system installation.

The default section alignment action for ld on M68000 systems is to align the code (.text) and data (.data and .bss combined) separately on 512-byte boundaries. Since MMU requirements vary from system to system, this alignment is not always desirable. This version of ld provides a mechanism to allow the specification of different section alignments for each system, allowing you to align each section separately on *n*-byte boundaries, where *n* is a multiple of 512. The default section alignment action for ld on this system is to align the code (.text) at byte 0 and the data (.data and .bss combined) at the 4 megabyte boundary (byte 10487576).

When all input files have been processed (and if no override is provided), ld will search the list of library directories (as with the –l flag option) for a file named **default.ld**. If this file is found, it is processed as an ld instruction file (or *ifile*). The **default.ld** file should specify the required alignment as outlined below. If it does not exist, the default alignment action will be taken.

The **default.ld** file should appear as follows, with *<alignment>* replaced by the alignment requirement in bytes:

```
SECTIONS {
      .text : {}
      GROUP ALIGN(<alignment>) : {
                    .data : {}
                    .bss  : {}
                    }
      }
```

Note: This system requires a *data rounding* that is an even multiple of 1 megabyte (1 megabyte is the segment size).

For example, a **default.ld** file of the following form would provide the same alignment as the default (512-byte boundary):

```
SECTIONS {
      .text : {}
      GROUP ALIGN(512) : {
                    .data : {}
                    .bss  : {}
                    }
      }
```

To get alignment on 2K-byte boundaries, the following **default.ld** file should be specified:

```
SECTIONS {
      .text : {}
      GROUP ALIGN(2048) : {
                     .data : {}
                     .bss : {}
                     }
       }
```

Note that this system requires a *data rounding* that is an even multiple of 1 megabyte (1 megabyte is the segment size).

For more information about the format of **ld** instruction files or the meaning of the commands, see "ld – Link Editor" in the UniPlus+ *Programming Guide.*

FILES

/bin/ld
/lib/*
/usr/lib/*

a.out	default output file

SEE ALSO

as(1), cc(1), a.out(4), ar(4).

WARNINGS

Through its flag options and input directives, the common link editor gives you great flexibility; however, if you use the input directives, you must assume some added responsibilities. Input directives should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, you must not place any object at virtual address zero in the data space.
- When you call the link editor through cc(1), a startup routine is linked with your program. This routine calls exit( ) (see exit(2)) after execution of the main program. If you call the link editor directly, you must insure that the program always calls exit( ), rather than falling through the end of the entry routine.

```
GROUP ALIGN(2048) : {
                    .data : {}
                    .bss : {}
                    }
}
```

For more information about the format of *ld* instruction files or the meaning of the commands, see **LD** in the *Programming Guide*.

FILES

/lib
/usr/lib
a.out default output file

SEE ALSO

as(1), cc(1), a.out(4), ar(4).

WARNINGS

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.
- When the link editor is called through *cc*(1), a startup routine is linked with the user's program. This routine calls exit ( ) (see *exit*(2)) after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls exit( ) rather than falling through the end of the entry routine.

NAME

ld − link editor for common object files

SYNOPSIS

ld [**−eepsym**] [**−ffill**] [**−lx**] [**−m**] [**−ooutfile**] [**−r**] [**−s**] [**−usymname**] [**−z**] [**−F**] [**−Ldir**] [**−N**] [**−V**] [**−VSnum**] filenames

DESCRIPTION

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *ld* combines them, producing an object module that can either be executed or used as input for a subsequent *ld* run. The output of *ld* is left in *a.out*. This file is executable if no errors occur during the load. If any input file, *filename*, is not an object file, *ld* assumes it is either a text file containing link editor directives or an archive library.

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. The library (archive) symbol table (see *ar*(4)) is searched sequentially with as many passes as are necessary to resolve external references that can be satisfied by library members. Thus, the ordering of library members is unimportant.

The following options are recognized by *ld*.

−e*epsym*
Set the default entry point address for the output file to be that of the symbol *epsym*.

−f*fill* Set the default fill pattern for "holes" within an output section as well as initialized **bss** sections. The argument *fill* is a two-byte constant.

−l*x* Search a library **lib***x***.a**, where *x* is up to seven characters. A library is searched when its name is encountered, so the placement of a **−l** is significant. By default, libraries are located in /lib.

−m Produce a map or listing of the input/output sections on the standard output.

−o*outfile*
Produce an output object file by the name *outfile*. The name of the default object file is **a.out**.

−r Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor does not complain about unresolved references.

−s Strip line number entries and symbol table information from the output object file.

−u*symname*
Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−z Load the text segment at an offset from zero so that null pointer references generate a segmentation violation.

−F Create a demand-paged executable.

−L*dir* Change the algorithm of searching for **lib***x***.a** to look in *dir* before looking in /lib and /usr/lib. This option is effective only if it precedes the **−l**option on the command line.

−N Put the data section immediately following the text in the output file.

−V Output a message giving information about the version of *ld* being used.

−VS*num*
Use *num* as a decimal version stamp identifying the *a.out* file that is produced. The version stamp is stored in the optional header.

The following information about section alignment and MMU requirements should be considered at system installation.

The default section alignment action for *ld* on M68000 systems is to align the code (**.text**) and data (**.data** and **.bss** combined) separately on 512-byte boundaries. Since MMU requirements vary from system to system, this alignment is not always desirable. The version of *ld* for M68000 systems, therefore, provides a mechanism to allow the specification of different section alignments for each system.

When all input files have been processed (and if no override is provided), *ld* will search the list of library directories (as with the **−l** option) for a file named **default.ld**. If this file is found, it is processed as an *ld* instruction file (or **ifile**). The **default.ld** file should specify the required alignment as outlined below. If it does not exist, the default alignment action will be taken.

The **default.ld** file should appear as follows, with <*alignment*> replaced by the alignment requirement in bytes:

```
SECTIONS {
        .text : {}
        GROUP ALIGN(<alignment>) : {
                                .data : {}
                                .bss  : {}
                                }
        }
```

For example, a **default.ld** file of the following form would provide the same alignment as the default (512-byte boundary):

```
SECTIONS {
        .text : {}
        GROUP ALIGN(512) : {
                                .data : {}
                                .bss  : {}
                                }
        }
```

To get alignment on 2K-byte boundaries, the following **default.ld** file would be specified:

```
SECTIONS {
        .text : {}
        GROUP ALIGN(2048) : {
                                .data : {}
                                .bss : {}
                                }
        }
```

For more information about the format of *ld* instruction files or the meaning of the commands, see LD in the *Programming Guide*.

FILES

/lib	
/usr/lib	
a.out	default output file

SEE ALSO

as(1), cc(1), a.out(4), ar(4).

WARNINGS

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.
- When the link editor is called through *cc*(1), a startup routine is linked with the user's program. This routine calls exit ( ) (see *exit*(2)) after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls exit( ) rather than falling through the end of the entry routine.

NAME

ld5.0 − link editor

SYNOPSIS

ld5.0 [option] file ...

DESCRIPTION

Ld5.0 combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object files are given, and *ld5.0* combines them, producing an object module which can be either executed or become the input for a further *ld5.0* run. (In the latter case, the **−r** option must be given to preserve the relocation bits.) The output of *ld5.0* 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 beginning of the first routine.

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

The symbols "_etext", "_edata" and "_end" ("etext", "edata" and "end" in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld5.0 understands several options. Except for **−l**, they should appear before the file names.

−s "Strip" the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1).

−u Take the following argument as a symbol and enter it as 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*x* This option is an abbreviation for the library name "**/lib/5.0/lib***x***.a**", where *x* is a string. If that does not exist, *ld5.0* tries "**/usr/lib/5.0/lib***x***.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 *cc5.0*(1) to discard internally generated labels while retaining symbols local to routines.

−r Generate relocation bits in the output file so that it can be the subject of another *ld5.0* run. This flag also prevents final definitions from being given to common symbols, and suppresses the "undefined symbol" diagnostics.

−R *x* Set starting relocation address of program to *x* (*x* is in hex).

−LT *x* Set the text relocation address to *x* (*x* is in hex).

−LD *x* Set the data relocation address to *x* (*x* is in hex).

−LC *x* Set the common relocation address to *x* (*x* is in hex).

−LB *x* Set the bss relocation address to *x* (*x* is in hex).

−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 protection boundary following the end of the text.

−N *x* Set the data relocation boundary to *x* for shared text programs. The value *x* may be followed by a *k* or *K* to indicate multiplication by 1024.

−o The *name* argument after **−o** is used as the name of the *ld5.0* output file, instead of **a.out.**

−e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

−F *x* Add offset *x* to all data references (*x* is in hex).

EXAMPLE

```
ld5.0 -s /lib/5.0/crt0.o filea.o fileb.o -lc
```

will load subroutines "filea" with "fileb" for execution and remove its symbol table.

FILES

/lib/5.0/lib*.a	libraries
/usr/lib/5.0/lib*.a	more libraries
a.out	default output file
/lib/5.0/crt0.o	"C" start up routine

SEE ALSO

ar5.0(1), as5.0(1), cc5.0(1).

NAME

lex − generate programs for simple lexical tasks

SYNOPSIS

lex [**−rctvn**] [file] ...

DESCRIPTION

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

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are 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 new-line. Parentheses for grouping and vertical bar for alternation are also supported. 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. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character **$** at the end of an expression requires a trailing new-line. 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, 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+yyleng*. 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.

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, **−n** will not print out the − summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n*	number of positions is *n* (default 2000)
%n *n*	number of states is *n* (500)
%t *n*	number of parse tree nodes is *n* (1000)
%a *n*	number of transitions is *n* (3000)

The use of one or more of the above automatically implies the **−v** option, unless the **−n** option is used.

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;
                }
        }
```

SEE ALSO

yacc(1), malloc(3X).
LEX in the *Programming Tools Guide* .

BUGS

When given an illegal option, *id* reports the fact that it has been given an illegal option, but then continues to execute with the default options rather than stopping execution and printing a usage message.

NAME

line — read one line

SYNOPSIS

line

DESCRIPTION

Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

EXAMPLE

```
line
Hello world
```

will return

```
Hello world
```

In the Bourne shell (sh):

```
a='line'
hi there
echo $a
```

will return

```
hi there
```

In the C-shell (csh):

```
set a='line'
bye bye
echo $a
```

will return

```
bye bye
```

SEE ALSO

sh(1), read(2).

NAME

lint – a C program checker

SYNOPSIS

lint [options] file ...

DESCRIPTION

Lint attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **–c** or the **–o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

Lint will take all the **.c**,**.ln**, and **llib-l***x***.ln** (specified by **–l***x*) files and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the **–c** option is used, the **.ln** and the **llib-l***x***.ln** files are ignored.

Any number of *lint* options may be used, in any order, intermixed with filename arguments. The following options are used to suppress certain kinds of complaints:

–a Suppress complaints about assignments of long values to variables that are not long.

–b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints).

–h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

−u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program).

−v Suppress complaints about unused arguments in functions.

−x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

−l*x* Include additional lint library **llib-lx.ln**. For example, you can include a lint version of the Math Library **llib-lm.ln** by inserting **−lm** on the command line. This argument does not suppress the default use of **llib-lc.ln.** These lint libraries must be in the assumed directory. This option can be used to reference local *lint* libraries and is useful in the development of multi-file projects. To generate llib-lX.ln from llib-lX, use:

```
%cc −E −C −Dlint llib-lX | /usr/lib/lint/lint1 −vx −H/tmp/lint$$ > llib-1X.1n
% rm −f /tmp/lint$$
```

−n Do not check compatibility against either the standard or the portable lint library.

−p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

−c No longer available (see description for **-l**).

−o lib No longer available (see description for **-l**).

The **−D**, **−U**, and **−I** options of *cpp* (1) and the **−g** and **−O** options of *cc* (1) are also recognized as separate arguments. The **−g** and **−O** options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc* (1) command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/

at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like *exit*(2)).

/*VARARGS*n**/

suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/

turns on the **−v** option for the next function.

/*LINTLIBRARY*/

at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the **−v** and **−x** options.

Lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the **−c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **−c** and the **−o** options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the **−c** option. Each of these invocations produces a **.ln** file which corresponds to the **.c** file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the **−c** option), listing all the **.ln** files with the needed **−l***x* options. This will print all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

EXAMPLE

```
lint −b myfile.c
```

checks the consistency of the file "myfile.c". The **−b** option indicates that unreachable **break** statements are not to be checked. This option might well be used on files that *lex*(1) generates.

FILES

/usr/lib the directory where the lint libraries specified by the −l*x* option must exist

/usr/lib/lint[12] first and second passes

/usr/lib/llib-lc.ln declarations for C Library functions (binary format; source is in **/usr/lib/llib-lc)**

/usr/lib/llib-port.ln declarations for portable functions (binary format; source is in **/usr/lib/llib-port)**

/usr/lib/llib-lm.ln declarations for Math Library functions (binary format; source is in **/usr/lib/llib-lm**)

/usr/tmp/*lint* temporaries

SEE ALSO

cc(1), cpp(1), make(1).

BUGS

exit(2), *longjmp*(3C), and other functions that do not return are not understood; this causes various lies.

NAME

ln – make links

SYNOPSIS

ln [–s] *name1* [*name2*]

ln *name*... *directory*

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it.

There are two kinds of links: hard links and symbolic links. By default **ln** makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

The –s flag option causes **ln** to create symbolic links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open*(2) operation is performed on the link. A *stat*(2) on a symbolic link will return the linked-to file; an *lstat*(2) must be done to obtain information about the link. The *readlink*(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

Given one or two arguments, *ln* creates a link to an existing file *name1*. If *name2* is given, the link has that name; *name2* may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of *name1*.

Given more than two arguments, *ln* makes links to all the named files in the named directory. The links made will have the same name as the files being linked to.

FILES

/bin/ln

SEE ALSO

cp(1), mv(1), rm(1), link(2), lstat(2), readlink(2), stat(2), symlink(2).

NAME

login − sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file".

If *login* is invoked as a command, it must replace the initial command interpreter. This is accomplished by typing:

```
        exec login
```

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful *login.*

If you do not complete the *login* successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful *login*, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and the file **.profile** in the working directory is executed, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is − followed by the last component of the interpreter's pathname (i.e., **−sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used. If this field is "*", then a *chroot*(2) is done to the directory named in the directory field of the entry. At that point *login* is re-executed at the new level which must have its own root structure, including **/etc/login** and **/etc/passwd**.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your *login* name. The arguments may take either the form *xxx* or *xxx*=*yyy*. Arguments without an equal sign are placed in the environment as

L *n*=**xxx**

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

EXAMPLE

At the beginning of each terminal session, the following sort of message is displayed on the screen:

```
UniSoft 68000 UNIX
:login:
```

to which a user name is the appropriate response.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	systemwide personal profile (*sh*(1))
/etc/cshrc	systemwide personal csh startup (*csh*(1))
.profile	personal profile (*sh*(1))
.login	personal csh startup used at login time (*csh*(1))
.cshrc	personal csh startup (*csh*(1))
.logout	personal csh logout used at logout time (*csh*(1))

SEE ALSO

mail(1), newgrp(1), sh(1), su(1), passwd(4), profile(4), environ(5).

DIAGNOSTICS

Login incorrect

if the user name or the password cannot be matched.

No shell, cannot open password file, or *no directory*

consult a UNIX system programming counselor.

No utmp entry. You must exec "login" from the lowest level "sh".

if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname — get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable $LOGNAME, which is set when a user logs into the system.

EXAMPLE

```
logname
```

displays the $LOGNAME of the user logged into the system on the current port.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3X), environ(5), printenv(1).

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar* (1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort* (1) to find an ordering of a library suitable for one-pass access by *ld* (1). Note that the link editor *ld* (1) is capable of multiple passes over an archive in the portable archive format (see *ar* (4)) and does not require that *lorder* (1) be used when building an archive. The usage of the *lorder* (1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

EXAMPLE

```
ar cr library  lorder *.o | tsort
```

builds a new library from existing .o files.

FILES

*symref, *symdef   temporary files

SEE ALSO

ar(1), ld(1), tsort(1), ar(4).

BUGS

Object files whose names do not end with .o, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

lorder5.0 – find ordering relation for an object library

SYNOPSIS

lorder5.0 file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar5.0*(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld5.0*(1). Note that the link editor *ld5.0*(1) is capable of multiple passes over an archive in the portable archive format (see *ar5.0*(4)) and does not require that *lorder5.0*(1) be used when building an archive. The usage of the *lorder5.0*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

EXAMPLE

```
ar5.0 cr library  lorder5.0 *.o | tsort
```

builds a new library from existing .o files.

FILES

*symref, *symdef   temporary files

SEE ALSO

ar5.0(1), ld5.0(1), tsort(1), ar5.0(4).

BUGS

Object files whose names do not end with .o, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

lp [**–c**] [**–d**dest] [**–m**] [**–n**number] [**–o**option] [**–s**] [**–t**title] [**–w**] files
cancel [ids] [printers]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name – stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see below) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

–c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the **–c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **–c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

–d *dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

–m Send mail (see *mail*(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n *number* Print *number* copies (default of 1) of the output.

−o *option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin* (1M).

−s Suppress messages from *lp* (1) such as "request id is ...".

−t *title* Print *title* on the banner page of the output.

−w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp* (1) command. The command line arguments may be either request *ids* (as returned by *lp* (1)) or *printer* names (for a complete list, use *lpstat* (1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request. A request will be cancelled only if *cancel* is invoked by its owner or the super-user.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1).
accept(1M), lpadmin(1M), lpsched(1M) in the *UniPlus⁺ Administrator's Manual*.

NAME

lpstat − print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
−u"user1, user2, user3"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat −o
```

prints the status of all output requests.

−a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

−c[*list*] Print class names and their members. *List* is a list of class names.

−d Print the system default destination for *lp*.

−o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

−p[*list*] Print the status of printers. *List* is a list of printer names.

−r Print the status of the LP request scheduler.

−s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

−t Print all status information.

−u[*list*] Print status of output requests for users. *List* is a list of login names.

−v[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls − list contents of directory

SYNOPSIS

ls [**−RadCxmlnogrtucpFbqisf**] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **−C** and **−x** options enable multi-column formats, and the **−m** option enables stream output format in which files are listed across the page, separated by commas. In order to determine output formats for the **−C**, **−x**, and **−m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

There are an unbelievable number of options:

−R Recursively list subdirectories encountered.

−a List all entries; usually entries whose names begin with a period (.) are not listed.

−d If an argument is a directory, list only its name (not its contents); often used with **−l** to get the status of a directory.

−C Multi-column output with entries sorted down the columns.

−x Multi-column output with entries sorted across rather than down the page.

−m Stream output format.

−l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.

−n The same as **−l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

−o The same as **−l**, except that the group is not printed.

−g The same as **−l**, except that the owner is not printed.

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−t Sort by time modified (latest first) instead of by name.

−u Use time of last access instead of last modification for sorting (with the **−t** option) or printing (with the **−l** option).

−c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**−t**) or printing (**−l**).

−p Put a slash (/) after each filename if that file is a directory.

−F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.

−b Force printing of non-graphic characters to be in the octal \ddd notation.

−q Force printing of non-graphic characters in file names as the character (?).

−i For each file, print the i-number in the first column of the report.

−s Give size in blocks, including indirect blocks, for each entry.

−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.

The mode printed under the **−l** option consists of 10 characters that are interpreted as follows:

The first character is:

d if the entry is a directory;
b if the entry is a block special file;
c if the entry is a character special file;
p if the entry is a fifo (a.k.a. ''named pipe'') special file;
− if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, ''execute'' permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
− if the indicated permission is *not* granted.

The group-execute permission character is given as s if the file has set-group-ID mode; likewise, the user-execute permission character is given as s if the file has set-user-ID mode. The last character of the mode (normally x or −) is t if the 1000 (octal) bit of the mode is on; see *chmod*(1) for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (S and T respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

EXAMPLE

```
ls -l /etc
```

will list all entries in **/etc** in long format.

FILES

/etc/passwd	to get user IDs for **ls -l** and **ls -o**.
/etc/group	to get group IDs for **ls -l** and **ls -g**.
/usr/lib/terminfo/*	to get terminal information.

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

m4 − macro processor

SYNOPSIS

m4 [options] [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 files, or if a file name is −, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

−e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.

−s Enable line sync output for the C preprocessor (#line ...)

−B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

−H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

−S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

−T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **−D** or **−U** flags:

−D*name*[=*val*]
Defines *name* to *val* or to null in *val*'s absence.

−U*name*
undefines *name*.

Macro calls have the form:

```
name(arg1,arg2, ..., argn)
```

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

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 **$***n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is like **$***, but each argument is quoted (with the current quotes).

undefine — removes the definition of the macro named in its argument.

defn — returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef — like *define*, but saves any previous definition.

popdef — removes current definition of its argument(s), exposing the previous one if any.

ifdef — if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on the UNIX System versions of *m4*.

shift — returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

changequote — change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., \`').

changecom — change left and right comment markers from the default **#** and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

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 new-line.

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.

decr returns the value of its argument decremented by 1.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~ ; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

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.

translit transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is identical to *include*, except that it says nothing if the file is inaccessible.

syscmd executes the UNIX System command given in the first argument. No value is returned.

sysval is the return code from the last call to *syscmd.*

maketemp fills in a string of XXXXX in its argument with the current process ID.

m4exit causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.

m4wrap argument 1 will be pushed back at final EOF; example: m4wrap(‘cleanup( )’)

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.

traceon with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

EXAMPLE

```
m4 file1 file2 > outputfile
```

will run the *m4* macro processor on the files "file1" and "file2", redirecting the output into "outputfile".

SEE ALSO

cc(1), cpp(1).
M4 in the *Programming Tools Guide*.

SEE *MACHID*

NAME

m68k, pdp11, u3b, u3b5, vax — provide truth value about your processor type

SYNOPSIS

m68k

pdp11

u3b

u3b5

vax

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

m68k	True if you are on a 68000.
pdp11	True if you are on a PDP-11/45 or PDP-11/70.
u3b	True if you are on a 3B 20S.
u3b5	True if you are on a 3B 5 computer.
vax	True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (non-zero) value. These commands are often used within *make*(1) makefiles and shell procedures to increase portability.

SEE ALSO

make(1), sh(1), test(1), true(1).

NAME

mail, rmail − send mail to users or read mail

SYNOPSIS

mail [**−epqr**] [**−f** file]

mail [**−t**] persons

rmail [**−t**] persons

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input to determine the disposition of the message:

<new-line>	Go on to next message.
+	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
−	Go back to previous message.
s [*files*]	Save message in the named *files* (**mbox** is default).
w [*files*]	Save message, without its header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons* (yourself is default).
q	Put undeleted mail back in the *mailfile* and stop.
EOT (control-d)	Same as **q**.
x	Put all mail back in the *mailfile* unchanged and stop.
!*command*	Escape to the shell to do *command.*
*	Print a command summary.

The optional arguments alter the printing of the mail:

−e causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

−p causes all mail to be printed without prompting for disposition.

−q causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.

−r causes messages to be printed in first-in, first-out order.

−f*file* causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a >. The **−t** option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of **dead.letter**.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will interpret that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**. *Mail* will not use *uucp* if the remote system is the local system name (i.e., localsystem!user).

The *mailfile* may be manipulated in two ways to alter the tunction of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment. In order for forwarding to work properly, the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

EXAMPLE

mail carolyn

accepts whatever message is typed up to an EOF. Carolyn will be notified that she has mail the next time she logs in.

If you want to read mail that has been sent to you, simply type

mail

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the *mailfile*
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), uucp(1C), write(1).

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

mailx − interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the system *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages remain in this file until forcibly removed.

On the command line, *options* start with a dash (−) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

−d Turn on debugging output. Neither particularly interesting nor recommended.

−e Test for presence of mail. *Mailx* prints nothing and exits with a successful return code if there is mail to read.

−f [*filename*] Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.

−F Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).

−h *number* The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.

−H Print header summary only.

−i Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).

−n Do not initialize from the system default *Mailx.rc* file.

−N Do not print initial header summary.

−r *address* Pass *address* to network delivery software. All tilde commands are disabled.

−s *subject* Set the Subject header field to *subject*.

−u *user* Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.

−U Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable.

When reading mail, *mailx* is in *command mode.* A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode.* If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables.* These are flags and valued parameters which are set and cleared via the **set** and **unset** commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the **alias** command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode,* print is assumed. In *input mode,* commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a 'current' message, marked by a '>' in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces, which may include:

n Message number **n**.
. The current message.
^ The first undeleted message.
$ The last message.
***** All messages.
n−m An inclusive range of message numbers.
user All messages from **user**.
/string All messages with **string** in the subject line (case ignored).
:*c* All messages of type *c*, where *c* is one of:

- **d** deleted messages
- **n** new messages
- **o** old messages
- **r** read messages
- **u** unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: !, Copy, edit, followup, Followup, hold, mail, preserve, reply, Reply, shell, and visual. Any errors in the start-up file cause the remaining lines in the file to be ignored.

COMMANDS

The following is a complete list of *mailx* commands:

!*shell-command*
: Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

# *comment* Null command (comment). This may be useful in *.mailrc* files.

= Print the current message number.

? Prints a summary of commands.

alias *alias name ...*
group *alias name ...*
: Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates *name ...*
: Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, alternates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]
chdir [*directory*]
: Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*
: Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.

Copy [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.

delete [*msglist*]
Delete messages from the *mailbox.* If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [*header-file* ...]
ignore [*header-file* ...]
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **Print** and **Type** commands override this command.

dp [*msglist*]
dt [*msglist*] Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a **delete** command followed by a **print** command.

echo *string* ...
Echo the given strings (like *echo*(1)).

edit [*msglist*]
Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed*(1).

exit
xit Exit from *mailx*, without changing the *mailbox.* No messages are saved in the *mbox* (see also **quit**).

file [*filename*]
folder [*filename*]
Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current *mailbox.*
%**user**	the *mailbox* for **user.**
#	the previous file.
&	the current *mbox.*

Default file is the current *mailbox.*

folders Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]
Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **Followup, Save,** and **Copy** commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]

Respond to the first message in the *msglist* sending the message to the author of each message in the *msglist.* The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **f**ollowup, Save, and Copy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]

Prints the header summary for the specified messages.

group *alias name* ...
alias *alias name* ...

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]

Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the **z** command.

help Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]

Holds the specified messages in the *mailbox.*

if *s*|*r*
mail-commands
else
mail-commands
endif Conditional execution, where *s* will execute following *mail-commands*, up to an else or endif, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore *header-file* ...
discard *header-file* ...

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The Print and Type commands override this command.

list Prints all commands available. No explanation is given.

mail *name* ...

Mail a message to the specified users.

mbox [*msglist*]

Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **exit** and **quit** commands.

next [*message*]

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]

Preserve the specified messages in the *mailbox*.

Print [*msglist*]
Type [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ignore** command.

print [*msglist*]
type [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]

Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a filename, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]
respond [*message*]

Reply to the specified message, including all other recipients of the message. If "record" is set to a filename, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the Copy, followup, and Followup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]
save [*msglist*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the exit and quit commands).

set
set *name*
set *name*= *string*
set *name*= *number*

Define a variable called *name*. The variable may be given a null, string, or numeric value. Set by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [*msglist*]

Print the size in characters of the specified messages.

source *filename*

Read commands from the given file and return to command mode.

top [*msglist*]

Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*]

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See exit and quit.

Type [*msglist*]
Print [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

type [*msglist*]

print [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

undelete [*msglist*]

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name* ...

Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version Prints the current version and release date.

visual [*msglist*]

Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the save command.

xit
exit Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

z[+|-] Scroll the header display forward or backward one screen—full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~!*shell-command*

Escape to the shell.

~. Simulate end of file (terminate message input).

~:*mail-command*
~_*mail-command*

Perform the command-level request. Valid only when sending a message while reading mail.

~? Print a summary of tilde escapes.

~A Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~a Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~b *name* ... Add the *names* to the blind carbon copy (Bcc) list.

~c *name* ... Add the *names* to the carbon copy (Cc) list.

~d Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [*msglist*] Forward the specified messages. The messages are inserted into the message, without alteration.

~h Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *string* Insert the value of the named variable into the text of the message. For example, ~A is equivalent to '~i Sign.'

~m [*msglist*]
Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p Print the message being entered.

~q Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter.* See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~r *filename*
~< *filename*
~< ! *shell-command*
Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string* ... Set the subject line to *string.*

~t *name* ... Add the given *names* to the To list.

~v Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w *filename* Write the partial message onto the given file, without the header.

~x Exit as with **~q** except the message is not saved in *dead.letter.*

~| *shell-command*
Pipe the body of the message through the given *shell-command.* If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx.*

HOME=*directory*
The user's base of operations.

MAILRC=*filename*
The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **unset** command may be used to erase variables.

allnet All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the "metoo" variable.

append Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**

askcc Prompt for the Cc list after message is entered. Default is **noaskcc.**

asksub Prompt for subject if it is not specified on the command line with the **−s** option. Enabled by default.

autoprint Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint.**

bang Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang.**

cmd=*shell-command*
Set the default command for the **pipe** command. No default value.

conv=*conversion*
Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also

"sendmail" and the −U command line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (*pg*(1) by default). Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is $HOME/dead.letter.

debug Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*
The command to run when the edit or ˜e command is used. Default is *ed*(1).

escape=*c* Substitute *c* for the ˜ escape character.

folder=*directory*
The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header Enable printing of the header summary when entering *mailx*. Enabled by default.

hold Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the ˜. command. Default is **noignoreeof**. See also "dot" above.

keep When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*
The name of the file to save messages which have been read. The xit command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo If your login appears as a recipient, do not delete it from the list. Default is **nometoo.**

LISTER= *shell-command*
The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the pathname is absolute. Default is **nooutfolder**. See "folder" above and the Save, Copy, followup, and Followup commands.

page Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage.**

PAGER= *shell-command*
The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*
Set the *command mode* prompt to *string*. Default is "? ".

quiet Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet.**

record=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen= *number*
Sets the number of lines in a screen—full of headers for the headers command.

sendmail= *shell-command*
Alternate command for delivering messages. Default is

mail(1).

sendwait Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL = *shell-command*
The name of a preferred command interpreter. Default is *sh*(1).

showto When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign = *string*
The variable inserted into the text of a message when the **~a** (autograph) command is given. No default (see also **~i** (TILDE ESCAPES)).

Sign = *string*
The variable inserted into the text of a message when the **~A** command is given. No default (see also **~i** (TILDE ESCAPES)).

toplines = *number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL = *shell-command*
The name of a preferred screen editor. Default is *vi*(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

mail(1), pg(1), ls(1).

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail*(1) (the standard mail delivery program).

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [**–f** makefile] [**–p**] [**–i**] [**–k**] [**–s**] [**–r**] [**–n**] [**–b**] [**–e**] [**–m**] [**–t**] [**–d**] [**–q**] [names]

DESCRIPTION

The following is a brief description of all options and some special names:

–f *makefile* Description file name. *Makefile* is assumed to be the name of a description file. A file name of – denotes the standard input. The contents of *makefile* override the built-in rules if they are present.

–p Print out the complete set of macro definitions and target descriptions.

–i Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.

–k When a command returns nonzero status, abandon work on the current entry, but continue on other branches that do not depend on that entry.

–s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.

–r Do not use the built-in rules.

–n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

–b Compatibility mode for old makefiles. This option is on by default.

–e Environment variables override assignments within makefiles.

–m Print a memory map showing text, data, and stack. This option is a no-operation on systems without the *getu* system call.

–t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

–d Debug mode. Print out detailed information on files and times examined.

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [**–b**] [**–d**] [**–e**] [**–f** *makefile*] [**–i**] [**–k**] [**–m**] [**–n**] [**–p**] [**–q**] [**–r**] [**–s**] [**–t**] [*names*]

DESCRIPTION

The following is a brief description of all flag options and some special names:

–f *makefile* Description file name. **makefile** is assumed to be the name of a description file. A filename of – denotes the standard input. The contents of *makefile* override the built-in rules if they are present.

–p Print out the complete set of macro definitions and target descriptions.

–i Ignore error codes returned by invoked commands. This mode is entered if the fake target name .IGNORE appears in the description file.

–k When a command returns nonzero status, abandon work on the current entry, but continue on other branches that do not depend on that entry.

–s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.

–r Do not use the built-in rules.

–n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

–b Compatibility mode for old makefiles. This mode on by default.

–e Environment variables override assignments within makefiles.

–t **touch**(1) the target files (causing them to be up-to-date) rather than issue the usual commands.

–d Debug mode. Print out detailed information on files and times examined.

–q Question. The **make** command returns a zero or nonzero status code depending on whether the target file is or is not up-to-date.

.DEFAULT

If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS

Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT Same effect as the **−s** flag option.

.IGNORE Same effect as the **−i** flag option.

make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **−f** flag option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **−**, the standard input is taken. More than one **−** *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, nonnull list of targets, then a **:**, then a (possibly null) list of prerequisite files or dependencies. Text following a **;** and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or **#** begins a new dependency or macro definition. Shell commands may be continued across lines with the **** newline sequence. Everything printed by **make** (except the initial tab) is passed directly to the shell as is. Thus,

```
    echo  a\
    b
```

will produce

```
    ab
```

exactly the same as the shell would.

Sharp (**#**) and a newline character surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
    pgm:  a.o  b.o
        cc  a.o  b.o  −o  pgm
    a.o:  incl.h  a.c
        cc  −c  a.c
    b.o:  incl.h  b.c
        cc  −c  b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command may be the following: **−**, **@**, **−@**, or **@−**. If **@** is present, printing of the command is suppressed. If **−** is present, **make** ignores an error. A line is printed when it is executed unless the **−s** flag option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a **@**. The **−n** flag option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under "Environment," below). The **−t** (touch) flag option updates the modified date of a file without executing any commands.

Commands returning nonzero status normally terminate **make**. If the **−i** flag option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains **−**, the error is ignored. If the **−k** flag option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **−b** flag option allows old makefiles (those written for the old version of **make**) to run without errors. The difference between the old version of make and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of **make** assumed, if no command was specified explicitly, that the command was null.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS**.

Environment

The environment is read by **make**. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **−e** flag option causes the environment to override the macro assignments in a makefile.

The **MAKEFLAGS** environment variable is processed by **make** as containing any legal input flag option (except **−f**, **−p**, and **−d**) defined for the command line. Further, upon invocation, **make** "invents" the variable if it is not in the environment, puts the current flag options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input flag options. This proves very useful for "super-makes". In fact, as noted above, when the **−n** flag option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make −n** recursively on a whole software system to see what would have been executed. This is because the **−n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form *string1 = string2* are macro definitions. *string2* is defined as all characters up to a comment character or an unescaped newline. Subsequent appearances of $(*string1* [:*subst1* =[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1* =*subst2* is a substitute sequence. If it is specified, all nonoverlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, newline characters, and beginnings of lines. An example of the use of the substitute sequence is shown under "Libraries," below.

Internal Macros

There are five internally-maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is evaluated only for inference rules or the **.DEFAULT** rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent filename). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
    cc -c -O $*.c
```

or:

```
.c.o:
    cc -c -O $<
```

$?

The **$?** macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.

$%

The **$%** macro is only evaluated when the target is an archive library member of the form lib(file.o). In this case, **$@** evaluates to lib and **$%** evaluates to the library member, file.o.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, **make** has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c  .c~  .sh  .sh~  .c.o  .c~.o  .c~.c  .s.o  .s~.o
 .y.o  .y~.o  .l.o  .l~.o
.y.c  .y~.c  .l.c  .c.a  .c~.a  .s~.a  .h~.h
```

The internal rules for make are contained in the source file rules.c for the make program. These rules can be locally modified. To print out the rules compiled into the make on any machine in a form suitable for recompilation, the following command is used (works for the Bourne shell, sh(1), and Korn shell, ksh(1), only):

```
    make  -fp  -  2>/dev/null  </dev/null
```

The only peculiarity in this output is the (null) string which printf(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see sccsfile(4)). Thus, the rule .c~.o would transform an SCCS C source file into an object file (.o). Because the s. of the SCCS files is a prefix, it is incompatible with make's suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., .c:) is the definition of how to build *x* from *x*.c. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for .SUFFIXES. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

```
    .SUFFIXES:  .o  .c  .y  .l  .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; .SUFFIXES: with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
    pgm: a.o b.o
        cc a.o b.o -o pgm
    a.o b.o: incl.h
```

This is because make has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, CFLAGS, LFLAGS, and YFLAGS are used for compiler options to cc(1), lex(1), and yacc(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix .o from a file with suffix .c is specified as an entry with .c.o: as the target and no dependents. Shell commands associated with the target define the rule for making a .o file from a .c file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o.** (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form *.XX*.a where the *XX* is the suffix from which the archive member is to be made. An unfortunate by-product of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have lib(file.o) depend upon file.o explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:
     lib(file1.o) lib(file2.o) lib(file3.o)
     @echo lib is now up-to-date
.c.a:
     $(CC) -c $(CFLAGS) $<
     ar rv $@ $*.o
     rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into **make** and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:
     lib(file1.o) lib(file2.o) lib(file3.o)
     $(CC) -c $(CFLAGS) $(?:.o=.c)
     ar rv lib $?
     rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The $? list is defined to be the set of object file names (inside lib) whose C source files are out-of-date. The substitution mode translates the .o to .c . (Unfortunately, one cannot as yet transform to .c~; however, this may become possible in the future.) Note also, the disabling of the .c.a: rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

EXAMPLE

```
make CFLAGS=-O -f make.special
```

invokes **make** with command file **make.special** and redefines compiler options flag **CFLAGS** to be **-O**.

FILES

/bin/make
[M/m]akefile br s.[M/m]akefile

SEE ALSO

cc(1), cd(1), csh(1), ksh(1), lex(1), sh(1), touch(1), yacc(1), printf(3S), sccsfile(4).

BUGS

Some commands return nonzero status inappropriately; use **−i** to overcome the difficulty.

Filenames with the characters = @ : will not work.

Commands that are directly executed by the shell, notably cd(1), are ineffectual across newlines in **make**.

The syntax (lib(file1.o file2.o file3.o) is illegal.

You cannot build lib(file.o) from file.o.

The macro $(a:.o=.c~) does not work.

BUGS

Some commands return non-zero status inappropriately; use –i to overcome the difficulty. File names with the characters = @ : will not work. Commands that are directly executed by the shell, *notably* *cd*(1), are ineffectual across newlines in *make*. The syntax (lib(file1.o file2.o file3.o) is illegal. You cannot build lib(file.o) from file.o. The macro $(a:.o=.c~) does not work.

NAME

makekey − generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

Makekey is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output will be pipes.

EXAMPLE

```
/usr/lib/makekey
abcdefgh23
23xq5GyrhLTCA
```

The first line invokes *makekey*, the second line is the input to *makekey*, and the third is the new key generated by *makekey*.

SEE ALSO

crypt(1), ed(1), passwd(4).

NOTE

Not included in international distributions.

NAME

man − print entries in this manual

SYNOPSIS

man [options] [section] titles

DESCRIPTION

Man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are:

−T*term* Print the entry as appropriate for terminal type *term*. For a list of recognized values of *term*, type **help term2**. The default value of *term* is **450**.

−w Print on the standard output only the *path names* of the entries, relative to **/usr/catman**, or to the current directory for **−d** option.

−d Search the current directory rather than **/usr/catman**; requires the full file name (e.g., **cu.1c**, rather than just **cu**).

−c Causes *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, and **X**.

Man examines the environment variable **$TERM** (see *environ*(5)) and attempts to select options that adapt the output to the terminal being used. The **−T***term* option overrides the value of **$TERM**; in particular, one should use **−Tlp** when sending the output of *man* to a line printer.

Section may be changed before each *title*.

As an example:

```
man man
```

would reproduce on the terminal this entry, as well as any other entries named *man* that may exist in other sections of the manual.

FILES

/usr/catman/?_man/man[1-8]/* Preformatted manual entries

SEE ALSO

term(5).

CAVEAT

The *man* command prints manual entries that were formatted by *nroff* when the UNIX system was installed. Entries are originally formatted with terminal type 37, and are printed using the correct terminal filters as derived from the **−T***term* and **$TERM** settings. Typesetting or other non-standard printing of manual entries is not yet supported.

NAME

mc68cc − C compiler

SYNOPSIS

mc68cc [options] ... files ...

DESCRIPTION

The *mc68cc* command is the C compiler. It generates assembly instructions. *Mc68cc* accepts the following 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, no **.o** is produced. In the same way, arguments whose names end with **.s** are taken to be assembly source programs and are assembled to produce a **.o** file.

The following flags are interpreted by *mc68cc*. See *ld*(1) for link editor options and *as*(1) for assembler options.

−c Suppress the link-editing 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 link editing takes place, replace the standard startoff routine by one which automatically calls *monitor*(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program.

−f*sky* Use the sky floating point library.

−g Cause the compiler to generate additional information needed for the use of *sdb*(1).

−O Invoke an object-code optimizer. The optimizer moves, merges, and deletes code, so symbolic debugging with line numbers could be confusing when the optimizer is used.

−W*c,arg1[,arg2...]*

Hand off the argument[s] *argi* to pass *c*, where *c* is one of **[p012al]** indicating preprocessor, compiler first pass, compiler second pass, optimizer, assembler, or link editor, respectively. For example:

−W*a,-m*

Invoke the **m4** macro preprocessor on the input to the assembler. This must be done for a source file that contains assembler escapes.

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

−E Run only *cpp*(1) on the named C programs, and send the result to the standard output.

−P Run only *cpp*(1) on the named C programs, and leave the result on corresponding files suffixed **.i**.

−D*symbol*

Define *symbol* to the preprocessor. This mechanism is useful with the conditional statements in the preprocessor by allowing symbols to be defined external to the source file.

−U*symbol*

Undefine *symbol* to the preprocessor.

−I*dir* Change the algorithm for searching for *#include* files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, *#include* files whose names are enclosed in double quotes are searched for first in the directory of the *file* argument, then in directories named in *-I* options, and last in directories on a standard list. For *#include* files whose names are enclosed in <>, the directory of the *file* argument is not searched.

−B*string*

Construct pathnames for substitute preprocessor, compiler, assembler, and link editor passes by concatenating *string* with the suffixes **cpp**, **comp**, **optim**, **as**, and **ld**. If *string* is empty it is taken to be **/lib/o**.

Other arguments are taken to be either link editor option arguments or C-compatible object programs, typically produced by an earlier *mc68cc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are link-edited (in the order given) to produce an executable program with the name **a.out** unless the **-o** option of the link editor is used.

The C language standard was extended after UNIX 5.0 to allow arbitrary length variable names. This standard is supported on the M68000 family of processors. The **-T** option causes *mc68cc* to truncate variable names to provide backward compatibility with earlier systems.

FILES

file.c	input file
file.o	object file
file.s	assembly language file
a.out	link-edited output
/usr/tmp/mc68?	temporary
lib/cpp	preprocessor
lib/ccom	compiler
lib/optim	optimizer
bin/as	assembler, *as*(1)
bin/ld	link editor, *ld*(1)
/lib/libc.a	standard library, see (3)
/usr/lib/default.ld	loader command file for shared text programs
/lib/libsky.a	sky floating point routines
/lib/crt0sky.o	runtime startoff using sky
/lib/mcrt0sky.o	runtime startoff for profiling using sky

SEE ALSO

as(1), dis(1), ld(1).
"The C Programming Language" by B. W. Kernighan and D. M. Ritchie,

Prentice—Hall, 1978.
Programming Guide.

DIAGNOSTICS

The diagnostics produced by the C compiler are sometimes cryptic. Occasional messages may be produced by the assembler or link editor.

WARNING

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value are to explicitly call *exit*(2) or to leave the function **main()** with a *return expression;* construct.

NOTE

The compiler looks for the environment variable **M68000**, to determine whether or not stack growth code is to be generated. If this variable has the value **STACKCHECK** then stack growth code is produced. This environment variable is normally initialized by one of the system files when going into multiuser mode. If the host cpu is a **68000** then stack growth code is required.

The compiler produces shared text programs by default. In order to produce nonshared text programs, the **—N** loader flag and appropriate loader command file must be used.

NAME

mesg — permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

EXAMPLE

```
mesg y
```

changes the permission to "yes", and the system reports:

```
Is Yes; Was No
```

or whatever is the current and former state of your message permission.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir — make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically. These and other directories beginning with . are not visible in listings unless you use the -a option to *ls*.

Mkdir requires write permission in the parent directory.

EXAMPLE

```
mkdir letters
```

creates a directory **letters** as a subdirectory of the directory you are in at the time you employ the command.

SEE ALSO

rm(1), sh(1), umask(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

NAME

mkstr – create an error message file by massaging C source

SYNOPSIS

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name.

To process the error messages in the source to the message file *mkstr* keys on the string '**error("**' in the input stream. Each time it occurs, the C string starting at the " is placed in the message file followed by a new-line character and a null character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, i.e.:

```
char  efilname[] = "/usr/lib/pi_strings";
int    efil = –1;
```

You have to write the error-handling function yourself. The following is an example:

```
error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) < 0L | | read(efil, buf, 256) <= 0)
```

```
                        goto oops;
                printf(buf, a2, a3, a4);
        }
```

The optional – causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

EXAMPLE

If the current directory has files "a.c" and "b.c", then

```
mkstr exs x *.c
```

would create a new file "exs" which holds all the error messages extracted from the source files "a.c" and "b.c", as well as two new source files "xa.c" and "xb.c" which no longer contains the extracted error messages.

SEE ALSO

lseek(2).

BUGS

All the arguments except the name of the file to be processed are unnecessary.

AUTHORS

Bill Joy and Charles Haley.

NAME

mm, osdd, checkmm − print/check documents formatted with the MM macros

SYNOPSIS

mm [options] [files]

osdd [options] [files]

checkmm [files]

DESCRIPTION

Mm can be used to type out documents using *nroff* and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn* (see *eqn*(1)) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff* and MM are generated, depending on the options selected.

Osdd is equivalent to the command **mm −mosd**. For more information about the OSDD adapter macro package, see *mosd*(5).

Options for *mm* are given below. Any other arguments or flags (e.g., **−rC3**) are passed to *nroff* or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

−T*term* Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **$TERM** from the environment (see *profile*(4) and *environ*(5)) as the value of *term*, if **$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.

−12 Indicates that the document is to be produced in 12-pitch. May be used when **$TERM** is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 12 if this option is used.)

−c Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, and **X**.

−e Causes *mm* to invoke *neqn*; also causes *neqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).

−t Causes *mm* to invoke *tbl*(1).

−E Invokes the **−e** option of *nroff*.

−y Causes *mm* to use the non-compacted version of the macros (see *mm*(5)).

Checkmm is a program for checking the contents of the named *files* for errors in the use of the Memorandum Macros, missing or unbalanced *neqn* delimiters, and **.EQ/.EN** pairs. Note: The user need not use the *checkeq* program (see *eqn*(1)). Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

EXAMPLE

Assuming that the shell variable **$TERM** is set in the environment to **450**, the two command lines below are equivalent:

```
mm −t −rC3 −12 ghh*

tbl ghh* | nroff −cm −T450−12 −h −rC3
```

Mm reads the standard input when − is specified instead of any file names. (Mentioning other files together with − leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm −
```

HINTS

1. *Mm* invokes *nroff* with the **−h** flag. With this flag, *nroff* assumes that the terminal has tabs set every 8 character positions.
2. Use the **−o***list* option of *nroff* to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **−e**, **−t**, and **−** options, *together* with the **−o***list* option of *nroff* may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the **−s** option of *nroff* (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The **−s** option of *nroff* does not work with the **−c** option of *mm*, or if *mm* automatically invokes *col*(1) (see **−c** option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you'll get (often subtle) garbage; however, if you are redirecting output into a file, use the **−T37** option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

col(1), cw(1), env(1), eqn(1), greek(1), mmt(1), nroff(1), tbl(1), profile(4), mm(5), mosd(5), term(5).

DIAGNOSTICS

mm "mm: no input file" if none of the arguments is a readable file and *mm* is not used as a filter.

checkmm "Cannot open *filename*" if file(s) is unreadable. The remaining output of the program is diagnostic of the source file.

NAME

mmt, mvt − typeset documents, view graphs, and slides

SYNOPSIS

mmt [options] [files]

mvt [options] [files]

DESCRIPTION

These two commands are very similar to *mm*(1), except that they both typeset their input via *troff*(1), as opposed to formatting it via *nroff*; *mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by *tbl*(1) and/or *eqn*(1). The proper pipelines and the required arguments and flags for *troff*(1) and for the macro packages are generated, depending on the options selected.

Options are given below. Any other arguments or flags (e.g., **−rC3**) are passed to *troff*(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

−e Causes these commands to invoke *eqn*(1); also causes *eqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).
−t Causes these commands to invoke *tbl*(1).
−Tst Directs the output to the MH STARE facility.
−a Invokes the **−a** option of *troff*(1).
−y Causes *mmt* to use the non-compacted version of the macros (see *mm*(5)). No effect for *mvt*.

These commands read the standard input when **−** is specified instead of any file names.

Mvt is just a link to *mmt*.

HINT

Use the **−o***list* option of *troff*(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the **−e**, **−t**, and **−** options, *together* with the **−o***list* option of *troff*(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

EXAMPLE

```
        mmt -t -rC3 -12 -Tst file
```

is equivalent to

```
        tbl file | troff -cm -Tst -12 -h -rC3
```

SEE ALSO

env(1), eqn(1), mm(1), tbl(1), tc(1), troff(1), profile(4), environ(5), mm(5), mv(5).

DIAGNOSTICS

"m[mv]t: no input file" if none of the arguments is a readable file and the command is not used as a filter.

NAME

more – file perusal filter for crt viewing

SYNOPSIS

more [**–***n*] [**–c**] [**–d**] [**–f**] [**–l**] [**–s**] [**–u**] [*+linenumber* | *+/pattern*] [name ...]

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a CRT terminal. It normally pauses after each screenful, printing "--More--" at the bottom of the screen.

If the user then types a carriage return, one more line is displayed. If a carriage return is preceded by an integer, that number becomes the new window size. If the user hits a space, another screenful is displayed. If a space is preceded by an integer, that number of lines is printed. If the user hits **d** or control-D, 11 more lines (usually half a screenful) are displayed (a "scroll"). If a **d** or control-D is preceded by an integer, that number becomes the new scroll size.

More looks in the user's environment and, if necessary, the file **/etc/termcap** to determine terminal characteristics and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More also looks in the environment variable MORE to preset any options desired. For example, if you prefer to view files using the **–c** option, the *csh*(1) command

```
setenv MORE -c
```

or the *sh*(1) command sequence

```
MORE=-c; export MORE
```

will cause all subsequent invocations of *more* to use this **–c** as default. Normally, the user will place the command sequence which sets up the MORE environment in the shell start-up file, i.e. *.login* or *.profile.*

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the "--More--" prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

The following options are available:

–*n* is an integer which is the size (in lines) of the window which *more* will use instead of the default.

−c *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

−d causes *more* to prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful.

−f causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

−l causes *more* not to treat control-L (form feed) specially. If this option is not given, *more* will pause after any line that contains a control-L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

−s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

−u Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The **−u** option suppresses this processing.

+linenumber
option causes *more* to start up at *linenumber*

+/pattern
causes *more* to start up two lines before the line containing the regular expression *pattern*.

Once inside *more*, other sequences may be typed when *more* pauses. The sequences and their effects are as follows (*i* is an optional integer argument, defaulting to 1) :

= display the current line number

v start up the editor *vi* at the current line

h help command; give a description of all the *more* commands

i:n skip to the *i*-th next file given in the command line (Skips to last file if *n* doesn't make sense.)

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, more skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q

exit from *more* (same as q or Q; the user's interrupt character has the same effect).

. repeat the previous command.

i z same as typing a space except that *i*, if present, becomes the new window size.

i s skip *i* lines and print a screenful of lines

i f skip *i* screenfuls and print a screenful of lines

i n search for the *i*-th occurrence of the last regular expression entered

q or Q

exit from *more* (the user's interrupt character has the same effect).

i/expr

search for the *i*-th occurrence of the regular expression *expr*. Terminated either by pressing RETURN or the ESCAPE key. If the input is a file (rather than a pipe), and there are less than *i* occurrences of *expr*, then the position in the file remains unchanged. If the input is a file, and there are at least *i* occurrences of *expr*, a screenful is displayed starting two lines before the place where the expression was found. If the input is a pipe, and there are less than *i* occurrences of *expr*, an error message is printed and *more* exits (because the entire input stream has been read). If the input is a pipe, and there are at least *i* occurrences of *expr*, a screenful is displayed starting on the line where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

CONTROL-L (^L)
redraw the screen.

!command
invoke a shell with *command*.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the "--More--(xx%)" message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control–\). *More* will stop sending output, and will display the usual "--More--" prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the "/" and "!" commands.

If the standard output is not a teletype, then *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

EXAMPLE

```
nroff –ms +2 doc.n | more
```

would show the *nroff* output on the terminal screen.

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

AUTHOR

Eric Shienbrood

NAME

mv – move or rename files

SYNOPSIS

mv [–i] [–f] [–] *file1 file2*

mv [–i] [–f] [–] *file ...directory*

DESCRIPTION

Mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place; if not, *mv* exits.

In the second form, one or more *files* (plain files or directories) are moved to the *directory* with their original file-names.

Mv refuses to move a file onto itself.

Flag options:

–i stands for interactive mode. Whenever a move is to supercede an existing file, the user is prompted by the name of the file followed by a question mark. If the user answers with a line starting with "**y**", the move continues. Any other reply prevents the move from occurring.

–f stands for force. This flag option overrides any mode restrictions or the **–i** flag option.

– means interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

FILES

/bin/mv

SEE ALSO

cp(1), ln(1).

BUGS

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

NAME

netstat – show network status

SYNOPSIS

netstat [–**Aahimnrst**] [–**p**protocol] [–**a**] [interval] [system] [core]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. The options have the following meaning:

–A show the address of any associated protocol control blocks; used for debugging

–a show the state of all sockets; normally sockets used by server processes are not shown

–h show the state of the IMP host table

–i show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown)

–m show statistics recorded by the memory management routines (the network manages a "private share" of memory)

–n show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically)

–p *proto*
show the state of sockets utilizing protocol *proto*; the protocol is specified symbolically, and may be any protocol listed in the file */etc/protocols*.

–r show the routing tables

–s show per-protocol statistics

–t show timer statistics (must be used in conjunction with **–i**.

The arguments, *system* and *core* allow substitutes for the defaults "/unix" and "/dev/kmem".

If an *interval* is specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol.

Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **−n** option is specified, the address is printe d in the Internet "dot format"; refer to *inet*(3N) for more information regarding this format. Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route then discard it. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

hosts(4N), networks(4N), protocols(4N), services(4N), trpt(8N)

BUGS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

NAME

newform − change the format of a text file

SYNOPSIS

newform [**−s**] [**−i**tabspec] [**−o**tabspec] [**−b**n] [**−e**n] [**−p**n] [**−a**n] [**−f**] [**−c**char] [**−l**n] [files]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **−s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like −e 15 −l60 will yield results different from −l 60 −e15. Options are applied to all *files* on the command line.

−i *tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be **−−**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to **−8**. A *tabspec* of **−0** expects no tabs; if any are found, they are treated as **−1**.

−o *tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **−i** *tabspec*. If no *tabspec* is given, *tabspec* defaults to **−8**. A *tabspec* of **−0** means that no spaces will be converted to tabs on output.

−l *n* Set the effective line length to *n* characters. If *n* is not entered, −l defaults to 72. The default line length without the −l option is 80 characters. Note that tabs and backspaces are considered to be one character (use −i to expand tabs to spaces).

−b *n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **−l** *n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **−b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform −l1 −b7 file-name
```

The **−l1** must be used to set the effective line length shorter than any existing line in the file so that the **−b** option is activated.

−e *n* Same as **−b** *n* except that characters are truncated from the end of the line.

−c *k* Change the prefix/append character to *k*. Default character for *k* is a space.

−p *n* Prefix *n* characters (see **−c** *k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

−a *n* Same as **−p** *n* except characters are appended to the end of a line.

−f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **−o** option. If no **−o** option is specified, the line which is printed will contain the default specification of **−8**.

−s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by an * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform −s −i −l −a −e file-name
```

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*Newform* was called with a bad option.
not −s format	There was no tab on one line.
can't open file	Self explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 − normal execution
1 − for any error

SEE ALSO

csplit(1), tabs(1), fspec(4).

BUGS

Newform normally only keeps track of printable characters; however, for the **−i** and **−o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

Newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **−i−−** or **−o−−**).

If the **−f** option is used, and the last **−o** option specified was **−o−−**, and was preceded by either a **−o−−** or a **−i−−**, the tab specification format line will be incorrect.

NAME

newgrp — log in to a new group

SYNOPSIS

newgrp [−] [group]

DESCRIPTION

Newgrp changes a user's group identification. The user remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp* , regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (**PS1**) other than **$** (default) and has not exported **PS1**. After an invocation of *newgrp* , successful or not, their **PS1** will now be set to the default prompt string **$**. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

If the first argument to *newgrp* is a −, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

EXAMPLE

```
newgrp grpnam
```

would set the user's group ID to that of the group named "grpnam".

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

news − print news items

SYNOPSIS

news [−a] [−n] [−s] [items]

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *News* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current".

The **−a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The **−n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The −s option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

EXAMPLE

```
        news
```

will print out all files in /usr/news that have not been read previously by the account owner.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5).

NAME

nice − run a command at low priority

SYNOPSIS

nice [−increment] command [arguments]

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed. The *nice* command built into the C shell is different from **/bin/nice**, which can be used by either shell.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **− −10**.

EXAMPLE

For the Bourne shell:

```
nice -10 date
```

would cause the program *date* to be processed at a priority lower than normal (0), i.e., at +10. In the C shell, the same is achieved by typing in

```
nice +10 date
```

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl − line numbering filter

SYNOPSIS

nl [**−h**type] [**−b**type] [**−f**type] [**−v**start#] [**−i**incr] [**−p**] [**−l**num] [**−s**sep] [**−w**width] [**−n**format] [**−d**delim] file

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

−b *type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p** *string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).

−h *type* Same as **−b** *type* except for header. Default *type* for logical page header is **n** (no lines numbered).

−f *type* Same as **−b** *type* except for footer. Default for logical page footer is **n** (no lines numbered).

−p Do not restart numbering at logical page delimiters.

−v *start#* *Start#* is the initial value used to number logical page lines. Default is **1**.

−i *incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

−s *sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

−w *width* *Width* is the number of characters to be used for the line number. Default *width* is **6**.

−n *format* *Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

−l *num* *Num* is the number of blank lines to be considered as one. For example, −l2 results in only the second adjacent blank being numbered (if the appropriate −**ha**, −**ba**, and/or −**fa** option is set). Default is **1**.

−**d** *xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the −**d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

```
nl −v10 −i10 −d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nm – print name list of common object file

SYNOPSIS

nm [–d] [–e] [–f] [–h] [–n] [–o] [–T] [–u] [–v] [–V] [–x] *filename*(s)

DESCRIPTION

The **nm** command displays the symbol table of each common object file *filename*. *Filename* may be a relocatable or absolute common object file, or it may be an archive of relocatable or absolute common object files. **nm** prints the following information for each symbol. Note that the object file must have been compiled with the **–g** flag option of the **cc**(1) command for there to be *type*, *size*, or *line* information.

name
The name of the symbol.

value
Its value expressed as an offset or an address depending on its storage class.

class
Its storage class.

tv If the symbol is accessed through a transfer vector, this field contains *tv*.

type Its type and derived type. If the symbol is an instance of a structure or a union, the structure or union tag is given following the type (e.g., *struct-tag*). If the symbol is an array, the array dimensions are given following the type (e.g., **char[n] [m]**).

size Its size in bytes, if available.

line The source line number at which it is defined, if available.

section
For storage classes static and external, the object file section containing the symbol (e.g., text, data, or bss).

The output of **nm** may be controlled using the following flag options:

–d Print the value and size of a symbol in decimal (the default).

–o Print the value and size of a symbol in octal instead of decimal.

–x Print the value and size of a symbol in hexadecimal instead of decimal.

–h Do not display the output header data.

–v Sort external symbols by value before they are printed.

–n Sort external symbols by name before they are printed.

–e Print only static and external symbols.

−f Produce full output. Redundant symbols (.test, .data, .bss), normally suppressed, are printed.

−u Print undefined symbols only.

−V Print the version of the **nm** command executing on the standard error output.

−T Truncate long names. By default, **nm** prints the entire name of the symbols listed. Since object files can have symbol names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow, forcing every column after the name to be misaligned. The **−T** flag option causes **nm** to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Flag options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm** *name* **−e −v** and **nm −ve** *name* print the static and external symbols in *name*, with external symbols sorted by value.

FILES

/bin/nm

/usr/tmp/nm?????? flag option.

SEE ALSO

as(1), **cc**(1), **ld**(1), **a.out**(4), **ar**(4).

WARNINGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **−v** and **−n** flag options should be used only in conjunction with the **−e**

DIAGNOSTICS

nm: *name***: cannot open**

name cannot be read.

nm: *name***: bad magic**

name is not an appropriate common object file.

nm: *name***: no symbols**

The symbols have been stripped from *name*.

nm: name: bad magic *Name* is not an appropriate common object file.

nm: name: no symbols The symbols have been stripped from *name*.

NAME

nm5.0 − print name list

SYNOPSIS

nm5.0 [**−gnoprsu**] [file ...]

DESCRIPTION

Nm5.0 prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in **a.out5.0** are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

−g Print only global (external) symbols.

−n Sort numerically rather than alphabetically.

−o Prepend file or archive element name to each output line rather than only once. This option can be used to make piping to *grep*(1) more meaningful.

−p Don't sort; print in symbol-table order.

−r Sort in reverse order.

−s Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next highest value). This difference is the value printed. This flag turns on **−g** and **−n** and turns off **−u** and **−p**.

−u Print only undefined symbols.

EXAMPLE

```
nm5.0
```

prints the symbol list of **a.out5.0** , the default output file for the C compiler.

SEE ALSO

ar5.0(1), a.out5.0(4), ar5.0(4).

NAME

nohup — run a command immune to hangups (*sh* only)

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* immune to terminate (EOT, control-D) signal from the controlling terminal. With *nohup*, the priority is automatically incremented by 5. *Nohup* should be used with processes running in background (with "&") in order to prevent it from responding to interrupts or stealing the input from the next person who logs in on the same terminal. In *csh*, processes run in background are automatically immune to hangups.

If output is not redirected by the user, both the standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

```
nohup nroff -ms docsfile | lpr
```

runs the *nroff* command shown, immune to hangups, quits, and interrupts.

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and commands lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
tbl ofile | eqn | nroff > nfile
```

FILES

nohup.out standard output and standard error file.

SEE ALSO

chmod(1), csh(1), nice(1), sh(1), nice(2), signal(2)

WARNINGS

nohup command1; command2	*nohup* applies only to *command1*
nohup (command1; command2)	is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio −o <list >/dev/rmt/1m&
while
nohup cpio −o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file *errors*.

NAME

nroff − format text

SYNOPSIS

nroff [options] [files]

DESCRIPTION

Nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers. Its capabilities are described in the *NROFF/TROFF* chapter of the *Document Processing Guide* cited below.

An argument consisting of a minus (−) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

−o *list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end. (See *BUGS* below.)

−n *N* Number first generated page *N*.

−s *N* Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). This option does not work if the output of *nroff* is piped through *col*(1). When *nroff* halts between pages, an ASCII **BEL** is sent to the terminal.

−r *aN* Set register *a* (which must have a one-character name) to *N*.

−i Read standard input after *files* are exhausted.

−q Invoke the simultaneous input-output mode of the **.rd** request.

−z Print only messages generated by **.tm** (terminal message) requests.

−m *name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

−c *name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

−k *name* Compact the macros used in this invocation of *nroff*, placing the output in files **[dt].***name* in the current directory.

−T *name* Prepare output for specified terminal. Known *name*s are **37** for the (default) TELETYPE® Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, and **2631** for the Hewlett Packard 2631 line printer.

−e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

−u *n* Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

EXAMPLE

```
nroff -o4,8-10 -T300S -mabc file1 file2
```

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named "file1" and "file2", specifies the output terminal as a DASI-300S, and invokes the macro package *abc*.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

col(1), cw(1), eqn(1), greek(1), mm(1), tbl(1), troff(1), mm(5).
NROFF/TROFF and *A TROFF Tutorial* in the *Document Processing Guide*.

BUGS

Nroff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff* generates may be off by one day from your idea of what the date is.

When *nroff* is used with the **−o** *list* option inside a pipeline (e.g., with one or more of *cw*(1), *eqn*(1), and *tbl*(1)), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

od – octal dump

SYNOPSIS

od [–bcdosx] [file] [[+]offset[.][b]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing or an unrecognized option is specified, **–o** is default. The meanings of the format options are:

–b Interpret bytes in octal.

–c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.

–d Interpret words in unsigned decimal.

–o Interpret words in octal.

–s Interpret words in signed decimal.

–x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

EXAMPLE

```
od -d file +2
```

produces an octal dump of "filea" divided up into 32-bit words expressed in decimal equivalents with the dump starting point offset by 2 octal bytes.

SEE ALSO

dump(1).

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [–**f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.z with the same access modes, access and modified dates, and owner as those of *name*. The -**f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each *.z* file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file is zero length;

the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called *name*.z already exists;
the .z file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* can not be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.z use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.z (without destroying *name*.z) use the command:

```
pcat name > nnn
```

Pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the .z) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

Unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.z (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the "unpacked" name already exists;
if the unpacked file cannot be created.

EXAMPLE

```
pack file1
```

will pack file "file1" into "file1.z" and removes "file1" if packing is successful.

SEE ALSO

cat(1).

NAME

passwd — change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the *login name*.

Ordinary users may change only the password which corresponds to their login *name*.

Passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has "aged" sufficiently. If "aging" is insufficient, the new password is rejected and *passwd* terminates; see *passwd*(4).

Assuming "aging" is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical, the cycle of prompting for the new password is repeated for at most two more times.

Passwords must meet the following requirements:

> Each password must have at least six characters. Only the first eight characters are significant.
>
> Each password must contain at least two alphabetic characters (uppercase or lowercase) and at least one numeric or special character.
>
> Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.
>
> New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

One whose effective user ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

EXAMPLE

```
passwd
```

will give the response

> Changing password for < *username* >

and will then prompt for your present password and for the new password (twice).

FILES

/etc/passwd

SEE ALSO

login(1), id(1), su(1), crypt(3C), passwd(4).

NAME

paste − merge same lines of several files or subsequent lines of one file

SYNOPSIS

```
paste file1 file2 ...
paste −d list file1 file2 ...
paste −s [−d list] file1 file2 ...
```

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if − is used in place of a file name.

The meanings of the options are:

−d Without this option, the new-line characters of each but the last file (or last line in case of the **−s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **−d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **−s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), \\ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use −*d*"\\\\").

−s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **−d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

− May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLE

```
ls | paste −d" " −
```

list directory in one column.

```
ls | paste − − − −
```

list directory in four columns.

```
paste −s −d"\t\n" file
```

combine pairs of lines into lines.

SEE ALSO

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for **−s** option, no more than 12 input files may be specified.

PCAT(1) **SEE** *PACK* **PCAT(1)**

PDP11(1) **SEE** *MACHID* **PDP11(1)**

NAME

pg – file perusal filter for soft-copy terminals

SYNOPSIS

pg [**–number**] [**–pstring**] [**–cefns**] [**+linenumber**] [**+/pattern/**] [**files** ...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. (The file name – and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

–number
An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

–p *string*
Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

–c Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

–e Causes *pg not* to pause at the end of each file.

–f Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *–f* option inhibits *pg* from splitting lines.

−n Normally, commands must be terminated by a *<newline>* character. This option causes an automatic end of command as soon as a command letter is entered.

−s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+linenumber
Start up at *linenumber*.

+/pattern/
Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)*<newline>* or *<blank>*
This causes one page to be displayed. The address is specified in pages.

(+1) **l**
With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L**
Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a <*newline*>, even if the *−n* option is specified.

i/pattern/

Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

*i^pattern^*
i?pattern?

Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

For most search commands, the trailing /, ^, or ? may be omitted, except when **m**, **b**, or **t** is appended to indicate the position where the desired line should appear on the screen.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

s *filename*

Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <*newline*>, even if the *−n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q**

Quit *pg*.

!*command*

Command is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a *<newline>*, even if the *–n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to BREAK, DEL, or control-c by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of *more* will find that the z and f commands are available, but they work as *more* commands and not as *pg* commands, particularly with respect to addresses.

FILES

787.sp 38u

/usr/lib/terminfo/*

Terminal information data base

/tmp/pg*

Temporary file when input is from a pipe

SEE ALSO

crypt(1), ed(1), grep(1), terminfo(4).

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options (e.g., *crypt*(1)), terminal settings may not be restored correctly.

NAME

pr − print files

SYNOPSIS

pr [options] [files]

DESCRIPTION

Pr prints the named files on the standard output. If *file* is −, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **−s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

The below *options* may appear singly or be combined in any order:

+*k* Begin printing with page *k* (default is 1).

−*k* Produce *k*-column output (default is 1). The options **−e** and **−i** are assumed for multi-column output. Also, the **−k** option *must* be used if the **−w** (column width) option is used.

−a Print multi-column output across the page.

−m Merge and print all files simultaneously, one per column (overrides the −*k*, and **−a** options).

−d Double-space the output.

−e *ck* Expand *input* tabs to character positions *k*+1, 2∗*k*+1, 3∗*k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

−i *ck* In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2∗*k*+1, 3∗*k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

−n *ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **−m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

−w *k* For multi-column output, set the width of a line to *k* character positions instead of the default 72 characters. This option *must* be used with the **−k** (number of columns) option.

−o *k* Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

−l *k* Set the length of a page to *k* lines (default is 66).

−h Use the next argument as the header to be printed instead of the file name.

−p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

−f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on failure to open files.

−t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

−s *c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLE

```
pr −3dh "file list" file1 file2
```

prints "file1" and "file2" as a double-spaced, three-column listing headed by "file list".

```
pr −e9 −t < file1 > file2
```

writes "file1" on "file2", expanding tabs to columns 10, 19, 28, 37, ...

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1).

NAME

printenv − print out the environment

SYNOPSIS

printenv [argument]

DESCRIPTION

Printenv takes an environment variable name as an argument and prints only the value of that variable. If no argument is given, it prints the values for the entire environment.

Examples of environment variable names are:

HOME	path name of user's home directory.
SHELL	the shell present at login.
PATH	search path for binary programs.
TERM	type of terminal used.
LOGNAME	the login name of the user.
TERMCAP	terminal capabilities string.
EXINIT	a startup list of commands read by *ex*, *edit* and *vi*.

EXAMPLE

```
printenv HOME
```

prints the path name of your home directory.

SEE ALSO

csh(1), sh(1), environ(5).

NAME

prof − display profile data

SYNOPSIS

prof [−tcan] [−ox] [−g] [−z] [−h] [−s] [−m mdata] [prog]

DESCRIPTION

Prof interprets the profile file produced by the *monitor*(3C) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with the profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t**, **c**, **a**, and **n** determine the type of sorting of the output lines:

−t Sort by decreasing percentage of total time (default).

−c Sort by decreasing number of calls.

−a Sort by increasing symbol address.

−n Sort lexically by symbol name.

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

−o Print each symbol address (in octal) along with the symbol name.

−x Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

−g Include non-global symbols (static functions).

−z Include all symbols in the profile range (see *monitor*(3C)), even if associated with zero number of calls and zero time.

−h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

−s Print a summary of several of the monitoring parameters and statistics on the standard error output.

−m mdata
Use file *mdata* instead of **mon.out** for profiling data.

For the number of calls to a function to be tallied, the **−p** option of *cc*(1) must have been given when the file containing the function was compiled. This option to the *cc* command also arranges for the object file to include a special profiling start-up function that calls *monitor*(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes the **mon.out** file to be written. Thus, only programs that call *exit*(2) or return from *main* cause the **mon.out** file to be produced.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

cc(1), nm(1), exit(2), profil(2), monitor(3C).

BUGS

There is a limit of 600 functions that may have call counters established during program execution. If this limit is exceeded, other data is overwritten and the **mon.out** file is corrupted. The number of call counters used is reported automatically by the *prof* command whenever the number exceeds 250.

NAME

prs − print an SCCS file

SYNOPSIS

prs [**−d**[dataspec]] [**−r**[SID]] [**−e**] [**−l**] [**−c**[date-time]] [**−a**] files

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

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

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

−d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

−r[*SID*] Used to specify the *S*CCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed. The format for the date is: mm/dd/yy [hh:mm:ss].

−e Requests information for all deltas created *earlier* than and including the delta designated via the **−r** keyletter or the date given by the **−c** option.

−l Requests information for all deltas created *later* than and including the delta designated via the **−r** keyletter or the date given by the **−c** option.

−c[*date-time*] Cutoff date-time, in the form: YY[MM[DD[HH[MM[SS]]]]]. Units omitted from the 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 in the form: "**−c77/2/2 9:22:25**".

−a Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, i.e., delta type = *D*, deltas. If the **−a** keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. A tab is specified by **\t** and carriage return/new-line is specified by **\n**. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S

:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLE

```
prs −d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs −d"Newest delta for pgm :M:: :I: Created :D: By :P:" −r s.file
```

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case*:

```
prs s.file
```

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the **−a** keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).
SCCS in the *Programming Tools Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps − report process status

SYNOPSIS

ps [options]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

−e Print information about all processes.

−d Print information about all processes, except process group leaders.

−a Print information about all processes, except process group leaders and processes not associated with a terminal.

−f Generate a *full* listing. (See below for meaning of columns in a full listing).

−l Generate a *long* listing. See below.

−c *corefile* Use the file *corefile* in place of **/dev/mem**.

−s *swapdev* Use the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*; a *swapdev* of **/dev/null** will cause the user block to be zeroed out.

−n *namelist* The argument will be taken as the name of an alternate *namelist* file in place of **/unix**.

−t *termlist* Restrict listing to data about the processes associated with the terminals given in *termlist*. The *termlist* may be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a command and/or one or more spaces. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., **tty04**) or if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).

−p *proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.

−u *uidlist* Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the **−f** option is used, in which case the login name will be printed.

−g *grplist* Restrict listing to data about processes whose process group leaders are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:
- 0 swapped;
- 1 in core;
- 2 system process;
- 4 locked-in core (e.g., for physical I/O);
- 10 being swapped;
- 20 being traced by another process;
- 40 another tracing flag;

S (l) The state of the process:
- 0 non-existent;
- S sleeping;
- W waiting;
- R running;
- I intermediate;
- Z terminated;
- T stopped;
- X growing.

UID (f,l) The user ID number of the process owner; the login name is printed under the **−f** option.

PID (all) The process ID of the process; it is possible to kill a process if you know this datum.

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

PRI (l) The priority of the process; higher numbers mean lower priority.

NI (l) Nice value; used in priority computation.

ADDR (l) The memory address of the process, if resident; otherwise, the disk address.

SZ (l) The size in blocks of the core image of the process.

WCHAN (l) The event for which the process is waiting or sleeping; if blank, the process is running.

STIME (f) Starting time of the process.

TTY (all) The controlling terminal for the process.

TIME (all) The cumulative execution time for the process.

CMD (all) The command name; the full command name and its arguments are printed under the **−f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **−f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **−f** option, is printed in square brackets.

EXAMPLE

```
ps −ef
```

displays information about all processes, with or without terminals.

FILES

/unix	system namelist
/dev/mem	memory
/dev/swap	the default swap device
/etc/passwd	supplies UID information
/etc/ps_data	internal data structure
/dev	searched to find terminal ("tty") names

SEE ALSO

acctcom(1), kill(1), nice(1).

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

NAME

ps – report process status

SYNOPSIS

ps [options]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

−e Print information about all processes.

−d Print information about all processes, except process group leaders.

−a Print information about all processes, except process group leaders and processes not associated with a terminal.

−f Generate a *full* listing. (See below for meaning of columns in a full listing).

−l Generate a *long* listing. See below.

−c*corefile* Use the file *corefile* in place of **/dev/mem**.

−s*swapdev* Use the file *swapdev* in place of **/dev/swap**. This is useful when examining a *corefile*; a *swapdev* of **/dev/null** will cause the user block to be zeroed out.

−n*namelist* The argument will be taken as the name of an alternate *namelist* file in place of **/unix**.

−t*termlist* Restrict listing to data about the processes associated with the terminals given in *termlist*. The *termlist* may be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a command and/or one or more spaces. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., **tty04**) or if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).

−p*proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.

−u*uidlist* Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the **−f** option is used, in which case the login name will be printed.

−g*grplist* Restrict listing to data about processes whose process group leaders are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; all means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:

0	swapped;
1	system process;
2	being traced by another process;
4	another tracing flag;
10	process cannot be woken by a signal;
20	in core;
40	locked in memory

S (l) The state of the process:

-	non-existent;
S	sleeping;
R	running;
I	intermediate;
Z	terminated;
T	stopped.

UID (f,l) The user ID number of the process owner; the login name is printed under the **−f** option.

PID (all) The process ID of the process; it is possible to kill a process if you know this datum.

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

PRI (l) The priority of the process; higher numbers mean lower priority.

NI (l) Nice value; used in priority computation.

ADDR (l) The memory address of the "u-area" (a pointer to the page tables) of the process, if resident; otherwise, the disk address.

SZ (l) The size in logical pages of the core image of the process.

WCHAN (l) The event for which the process is waiting or sleeping; if blank, the process is running.

STIME (f) Starting time of the process.

TTY (all) The controlling terminal for the process.

TIME (all) The cumulative execution time for the process.

CMD (all) The command name; the full command name and its arguments are printed under the **–f** option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

Under the **–f** option, *ps* tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the **–f** option, is printed in square brackets.

EXAMPLE

```
ps –ef
```

displays information about all processes, with or without terminals.

FILES

/unix	system namelist
/dev/mem	memory
/dev/swap	the default swap device
/etc/passwd	supplies UID information
/etc/ps_data	internal data structure
/dev	searched to find terminal ("tty") names

SEE ALSO

acctcom(1), kill(1), nice(1).

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

Processes which are swapped onto other than the default swap device (see *swap*(1M)) will have some invalid information printed out.

NAME

ptx − permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

Ptx generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. *Ptx* output is in the form:

```
.xx "tail" "before keyword" "keyword and after" "head"
```

where **.xx** is assumed to be an *nroff* or *troff* macro provided by the user, or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter.

−w *n* Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.

−g *n* Use the next argument, *n*, as the number of characters that *ptx* will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.

−o *only* Use as keywords only the words given in the *only* file.

−i *ignore* Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use **/usr/lib/eign** as the *ignore* file.

−b *break* Use the characters in the *break* file to separate words. Tab, new-line, and space characters are *always* used as break characters.

−r Take any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

EXAMPLE

```
If "file1" contains:  once upon a time
                      in the middle of a large
                      dark forest

     ptx file1
```

responds with:

```
.xx "" "" "dark forest" ""
.xx "" "dark" "forest" ""
.xx "" "in the middle of a" "large" ""
.xx "" "in the" "middle of a large" ""
.xx "" "" "once upon a time" ""
.xx "" "once" "upon a time" ""
```

FILES

/bin/sort
/usr/lib/eign
/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), troff(1), mm(5), mptx(5).

BUGS

Line length counts do not account for overstriking or proportional spacing. Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

NAME

put − puts a file onto a remote machine.

SYNOPSIS

put [**−p** *port*] [−sSPEED] [**−i** [ID]] *fromfile* [*tofile*]
put [**−p** *port*] [−sSPEED] **−c** *command* [*args*] ...

DESCRIPTION

Put is part of system of programs useful for transferring files between UNIX systems. It is the "uploader" designed to transmit files from a local machine to a remote machine. For a brief discussion of the take/put system and installation instructions, see the **TAKE/PUT** chapter of the *Administrator Guide.*

The default port is **/dev/tty0**; the **−p** option can be used to specify an alternate output port. The default speed is determined by the system; the −s option can be used to specify a speed. If *tofile* is unspecified, then it is assumed to be the same as *fromfile.* If *fromfile* is a directory, *tofile* must be a directory on the remote machine (or if nonexistent, the last existent directory specified in the pathname must be writable).

The **−i[ID]** option specifies a system ID and is the mechanism for remapping pathnames on the remote machine. The system ID is passed to the remote machine where it is used to generate pathname prefixes (using the **/etc/take_oem** file) which are appended to the *tofile* pathname supplied by *put.* If an ID is specified when using the **−i** option, it is used on the remote machine. If no ID is specified, the default ID is read from the **/etc/sys_id** file if it exists; if the **/etc/sys_id** does not exist, the system ID is considered to be the user name of the invoker of *put7* (i.e., the user who logged in over the port used).

The **−c** option is useful for executing an arbitrary command on the remote machine. All arguments following the **−c** flag are collected, transmitted to the remote machine and executed as a single command. The standard input to the *put* program is sent to the remote machine to become the standard input to the command specified. The standard error of the remote command becomes the standard error of *put.* The standard output of the remote command is not returned. The exit status of the remote command is returned as the exit status of *put.*

In order to perform its function, *put*(1C) interfaces with the program **/usr/bin/put7** on the remote machine.

EXAMPLE

```
put /a/b/c
```

puts the contents of the directory (or file) "/a/b/c" on the local machine into a similarly named directory (or file) on the remote machine; if "/a/b/c" did not previously exist on the remote machine, it is created; otherwise it is overwritten.

```
put file.c /x/y/z
```

puts the contents of "file.c" on the local machine into "/x/y/z/file.c" on the remote machine. Note that "file.c" is created on the remote machine if "z" is a directory; if "z" is a file rather than a directory, its contents are overwritten but its name remains "z" rather than becoming "file.c".

FILES

fromfile The local file name. When using the **−i** option, this file should be specified as a pathname starting at the root of the local machine.

tofile The remote file name; if *tofile* is null, *tofile* is defaulted to *fromfile*.

SEE ALSO

cu(1C), take(1C)
TAKE/PUT in the *Administrator Guide*.

NAME

pwd — working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the path name of the working (current) directory.

EXAMPLE

```
pwd
```

produces a pathname, such as **/usr/games**, indicating what directory you are currently in.

SEE ALSO

cd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX system programming counselor.

NAME

ratfor − rational Fortran dialect

SYNOPSIS

ratfor [options] [files]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

```
statement grouping:
        { statement; statement; statement }

decision-making:
        if (condition) statement [ else statement ]
        switch (integer value) {
                case integer:    statement
                ...
                [ default: ]     statement
        }

loops:
        while (condition) statement
        for (expression; condition; expression) statement
        do limits statement
        repeat statement [ until (condition) ]
        break
        next
```

and some syntactic sugar to make programs easier to read and write:

```
free form input:
        multiple statements/line; automatic continuation

comments:
        # this is a comment.

translation of relationals:
        >, >=, etc., become .GT., .GE., etc.

return expression to caller from function:
        return (expression)

define:
        define name replacement

include:
        include file
```

The option **−h** causes quoted strings to be turned into 27H constructs. The **−C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **−6x** makes the continuation character **x** and places it in column 6.

Ratfor is best used with *f77*(1).

SEE ALSO

efl(1), f77(1).
B. W. Kernighan and P. J. Plauger, "Software Tools", Addison-Wesley, 1976.

NAME

rcp − remote file copy

SYNOPSIS

rcp file1 file2
rcp [−r] file ... directory

DESCRIPTION

Rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form "rhost:path", or a local file name (containing no ':' characters, or a '/' before any ':'s.)

If the −r is specified and any of the source files are directories, *rcp* copies each subtree rooted at that name; in this case the destination must be a directory.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

Rcp does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via *remsh*(1N).

Rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form "rhost.rname" to use *rname* rather than the current user name on the remote host.

SEE ALSO

ftp(1N), remsh(1N), rlogin(1N)

BUGS

Doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.
Is confused by any output generated by commands in a .login, .profile, or .cshrc file on the remote host.

NAME

rcvhex — translates Motorola S-records from downloading into a file

SYNOPSIS

rcvhex [**−p** port] [**−c** command] file

DESCRIPTION

Rcvhex translates Motorola S-records shipped from a port into a file. The following options are available:

p *port* specifies an alternate port for reception; the default port is **/dev/tty0.**

c *command* ship the specified command (in quotes) over the remote port; the default is to not ship anything.

lfile File to be created by *rcvhex*.

The file's starting address must be zero and successive records must be sequential.

AUTHOR

Asa Romberger, UniSoft Systems

SEE *ED*

NAME

regcmp − regular expression compile

SYNOPSIS

regcmp [−] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp*(3X) 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 code. Compiled regular expressions are represented as **extern char** vectors. *File*.i files may thus be *included* into C programs, or *file*.c files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. 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 that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X).

NAME

remsh − remote shell

SYNOPSIS

remsh host [−lusername] [−**n**] command
host [−lusername] [−**n**] command

DESCRIPTION

Remsh connects to the specified *host*, and executes the specified *command*. *Remsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *remsh* normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the −l option. This remote name must be equivalent (in the sense of *rlogin*(1N)) to the originating account; no provision is made for specifying a password with a command.

If you omit *command*, then instead of executing a single command, you will be logged in on the remote host using *rlogin*(1N).

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

```
remsh otherhost cat remotefile >> localfile
```

appends the remote file *remotefile* to the localfile *localfile*, while

```
remsh otherhost cat remotefile ">>" otherremotefile
```

appends *remotefile* to *otherremotefile*.

Host names are given in the file /etc/hosts. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory /usr/hosts; if you put this directory in your search path then the **remsh** can be omitted.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1N)

BUGS

You cannot run an interactive command (like *vi*(1)); use *rlogin*(1N).

NAME

rlogin – remote login

SYNOPSIS

rlogin rhost [–ec] [–8] [–lusername]
rhost [–ec] [–8] [–lusername]

DESCRIPTION

Rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The hosts names must be the standard names as described in *remsh*(1N)). When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file .rhosts in his login directory. Each line in this file should contain a *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid security problems, the .rhosts file must be owned by either the remote user or root. Note that, for security reasons, root is an exception to the above; a superuser or an equivalent host must still provide the password to login as root to login as root unless the root account has its own private equivalence list in a .rhosts file in the root directory. A .rhosts file for a root account is not recommended where secure sytems are required.

Your remote terminal type is the same as your local terminal type (as given in your environment TERM variable). All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly. The optional argument **–8** allows an eight-bit data path, otherwise parity bits are stripped. A line of the format "~." disconnects from the remote host, where "~." is the escape character. A different escape character may be specified by the **–e** option. There is no space separating this option flag and the argument character.

SEE ALSO

remsh(1N)

FILES

/usr/hosts/*   for *rhost* version of the command

BUGS

More terminal characteristics should be propagated.

NAME

rm, rmdir − remove files or directories

SYNOPSIS

rm [**−fri**] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with **y** the file is deleted, otherwise the file remains. No questions are asked when the **−f** option is given or if the standard input is not a terminal. The **−f** option also prevents all error messages from being printed.

If a designated file is a directory, an error comment is printed unless the optional argument **−r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **−i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **−r**, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

EXAMPLE

```
rm -r dirname
```

will remove the entire contents of the named directory and all subdirectories, and finally the directory itself, with no questions asked.

SEE ALSO

unlink(2).

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file .. merely to avoid the antisocial consequences of inadvertently doing something like:

```
rm −r .*
```

NAME

rmdel − remove a delta from an SCCS file

SYNOPSIS

rmdel −rSID files

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (i.e., if a *p-file* (see *get*(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*).

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

The exact permissions necessary to remove a delta are documented in the **SCCS** chapter of the *Programming Tools Guide*. Simply stated, they are either: (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

EXAMPLE

```
rmdel -r1.2 s.test1.c
```

would remove the latest delta version (i.e., 1.2) for "s.test1.c".

FILES

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

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(4).
SCCS in the *Programming Tools Guide*.

DIAGNOSTICS

Use *help*(1) for explanations.

RMDIR (1) **SEE** *RM* **RMDIR (1)**

RSH (1) **SEE** *SH* **RSH (1)**

NAME

ruptime − show host status of local machines

SYNOPSIS

ruptime [−a] [−l] [−t] [−u]

DESCRIPTION

Ruptime gives a status line like *uptime* for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 5 minutes are shown as being down.

Users idle an hour or more are not counted unless the **−a** flag is given.

Normally, the listing is sorted by host name. The **−l** , **−t** , and **−u** flags specify sorting by load average, uptime, and number of users, respectively.

FILES

/usr/spool/rwho/whod.*	data files

SEE ALSO

rwho(1N)

NAME

rwho − who's logged in on local machines

SYNOPSIS

rwho [−a]

DESCRIPTION

The *rwho* command produces output similar to *who,* but for all machines on the local network. If no report has been received from a machine for 5 minutes then *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, then *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of *rwho* unless the **−a** flag is given.

FILES

/usr/spool/rwho/whod.*   information about other machines

SEE ALSO

ruptime(1N), rwhod(8N)

BUGS

This is unwieldy when the number of machines on the local net is large.

NAME

sact − print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the −e option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 specifies the SID for the new delta to be created.

Field 3 contains the logname of the user who will make the delta (i.e., executed a *get* for editing).

Field 4 contains the date that **get −e** was executed.

Field 5 contains the time that **get −e** was executed.

EXAMPLE

If the user has done a **get −e**, but not a **delta** to merge the new changes, doing a

```
sact s.test1.c
```

would show:

```
1.2 1.3 eryk 82/11/10 16:10:35
```

indicating that a new version numbered 1.3 is in the process of being made from version numbered 1.2 by user "eryk". The **get −e** for the file was done on 82/11/10 at 16:10:35.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sag − system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

Sag graphically displays the system activity data stored in a binary data file by a previous *sar*(1) run. Any of the *sar* data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. *Sag* invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what's available). These *options* are passed thru to *sar*:

−s *time* Select data later than *time* in the form hh[:mm]. Default is 08:00.

−e *time* Select data up to *time*. Default is 18:00.

−i *sec* Select data at intervals as close as possible to *sec* seconds.

−f *file* Use *file* as the data source for *sar*. Default is the current daily data file **/usr/adm/sa/sa***dd*.

Other *options*:

−T *term* Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. If *term* is **vpr**, output is processed by **vpr −p** and queued to a Versatec printer. Default for *term* is **$TERM**.

−x *spec* x axis specification with *spec* in the form:
"name [op name] ... [lo hi]"

−y *spec* y axis specification with *spec* in the same form as above.

Name is either a string that will match a column header in the *sar* report, with an optional device name in square brackets, e.g., **r+w/s [dsk−1]**, or an integer value. *Op* is +, −, *, or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and − have precedence over * and /. Evaluation is left to right. Thus A / A + B * 100 is evaluated (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be given for −y. Enclose the **−x** and **−y** arguments in "" if blanks or \<CR> are included. The **−y** default is:

−y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLE

```
sag
```

will show today's CPU utilization.

FILES

/usr/adm/sa/sa*dd* daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G).

NAME

sar − system activity reporter

SYNOPSIS

sar [−**ubdycwaqvmA**] [−**o** file] t [n]

sar [−**ubdycwaqvmA**] [−**s** time] [−**e** time] [−**i** sec] [−**f** file]

DESCRIPTION

Sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the −**o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, *sar* extracts data from a previously recorded *file*, either the one specified by −**f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa** *dd* for the current day *dd*. The starting and ending times of the report can be bounded via the −**s** and −**e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The −**i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

−**u** Report CPU utilization (the default):
%usr, %sys, %wio, %idle − portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

−**b** Report buffer activity:
bread/s, bwrit/s − transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s − accesses of system buffers;
%rcache, %wcache − cache hit ratios, e.g., 1 − bread/lread;
pread/s, pwrit/s − transfers via raw (physical) device mechanism.

−**d** Report activity for each block device, e.g., disk or tape drive:
%busy, avque − portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s − number of data transfers from or to device, number of bytes transferred in 512 byte units;
avwait, avserv − average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

−**y** Report TTY device activity:
rawch/s, canch/s, outch/s − input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s − receive, transmit and modem interrupt rates.

−**c** Report system calls:
scall/s − system calls of all types;
sread/s, swrit/s, fork/s, exec/s − specific system calls;
rchar/s, wchar/s − characters transferred by read and write system calls.

−**w** Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s − number of transfers and number of 512 byte units transferred for swapins (including initial loading of some programs) and swapouts;
pswch/s − process switches.

−a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

−q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc − run queue of processes in memory and runnable;
swpq-sz, %swpocc − swap queue of processes swapped out but ready to run.

−v Report status of text, process, inode and file tables:
text-sz, proc-sz, inod-sz, file-sz − entries/size for each table, evaluated once at sampling point;
text-ov, proc-ov, inod-ov, file-ov − overflows occurring between sampling points.

−m Report message and semaphore activities:
msg/s, sema/s − primitives per second.

−A Report all data. Equivalent to **−udqbwcayvm**.

EXAMPLE

```
sar
```

shows today's CPU activity so far.

```
sar −o temp 60 10
```

watches CPU activity evolve for 10 minutes and saves data.

```
sar −d −f temp
```

later reviews disk and tape activity from that period.

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G).
sar(1M) in the *UniPlus⁺ Administrator's Manual*.

NAME

sar − system activity reporter

SYNOPSIS

sar [**−ubycwaqvmA**] [−ofile] t [n]

sar [**−ubycwaqvmA**] [−stime] [−etime] [−isec] [−ffile]

DESCRIPTION

Sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the **−o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, *sar* extracts data from a previously recorded *file*, either the one specified by **−f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **−s** and **−e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **−i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

−u Report CPU utilization (the default):
%usr, %sys, %wio, %idle − portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

−b Report buffer activity:
bread/s, bwrit/s − transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s − accesses of system buffers;
%rcache, %wcache − cache hit ratios, e.g., 1 − bread/lread;
pread/s, pwrit/s − transfers via raw (physical) device mechanism.

−y Report TTY device activity:
rawch/s, canch/s, outch/s − input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s − receive, transmit and modem interrupt rates.

−c Report system calls:
scall/s − system calls of all types;
sread/s, swrit/s, fork/s, exec/s − specific system calls;
rchar/s, wchar/s − characters transferred by read and write system calls.

−w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s − number of transfers and number of 512 byte units transferred for swapins (including initial loading of some programs) and swapouts;
pswch/s − process switches.

−a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

−q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc − run queue of processes in memory and runnable;
swpq-sz, %swpocc − swap queue of processes swapped out but ready to run.

−v Report status of process, inode, file, file record lock and file record header tables:
proc-sz, inod-sz, file-sz, lock-sz, fhdr-sz − entries/size for each table, evaluated once at sampling point;
proc-ov, inod-ov, file-ov − overflows occurring between sampling points.

−m Report message and semaphore activities:
msg/s, sema/s − primitives per second.

−A Report all data. Equivalent to **−uqbwcayvm**.

EXAMPLE

```
sar
```

shows today's CPU activity so far.

```
sar  −o temp 60 10
```

watches CPU activity evolve for 10 minutes and saves data.

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G).
sar(1M) in the *UniPlus⁺ Administrator's Manual*.

NAME

sccsdiff − compare two versions of an SCCS file

SYNOPSIS

sccsdiff −rSID1 **−r**SID2 [**−p**] [**−sn**] files

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files. *Sccsdiff* first outputs lines resembling the ed commands to convert file1 into file2. It then outputs the actual lines that differ.

−r *SID ?* *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.

−p pipe output for each file through *pr*(1).

−s *n* *n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

EXAMPLE

```
sccsdiff −r1.1 −r1.2 s.test1.c
```

would show the differences between version 1.1 and version 1.2 of the file "test1.c".

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).
SCCS in the *Programming Tools Guide.*

DIAGNOSTICS

"*file: No differences*" If the two versions are the same.
Use *help*(1) for explanations.

NAME

script – make typescript of terminal session

SYNOPSIS

script [–a] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the –a option is given. It can be sent to the line printer later with **lp** . If no file name is given, the typescript is saved in the file *typescript*.

Note that **script** uses both standard input and standard output and that neither may be redirected via a pipe, < or >.

The script ends when the forked shell exits.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

BUGS

Script places everything in the log file. This is not what the naive user expects.

NAME

sdb − symbolic debugger

SYNOPSIS

sdb [−w] [−W] [objfil[corfil[directory]]]

DESCRIPTION

Sdb is a symbolic debugger which can be used with C and *f77* programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

Objfil is normally an executable program file which has been compiled with the **−g** (debug) option; if it has not been compiled with the **−g** option, or if it is not an executable file, the symbolic capabilities of *sdb* are limited, but the file can still be examined and the program debugged. 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**. The core file need not be present. A **−** in place of *corfil* forces *sdb* to ignore any core image file. Source files used in constructing *objfil* must be in *directory* to be located.

It is useful to know that at any time there is a *current line* and *current file*. If *corfil* exists then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main*(). The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfil* cannot be found, or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the **−W** flag.

Names of variables are written just as they are in C or *f77*. Variables local to a procedure may be accessed using the form *procedure***:***variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable***.***member*, pointers to structure members as *variable***−>***member*, and array elements as *variable***[***number***]**. Pointers may be dereferenced by using the form *pointer***[0]**. Combinations of these forms may also be used. *F77* common variables may be referenced by using the name of the common block instead of the structure name. Blank common variables may be named by the form **.***variable*. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* interprets a structure as a set of variables; thus, it displays the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable***[***number***][***number***]...**, or as *variable***[***number,number,...***]**. In place of *number*, the form *number***;***number* may be used to indicate a range of values, ***** may be used to indicate all legitimate values for that subscript, or

NAME

sdb – symbolic debugger

SYNOPSIS

sdb [**–w**] [**–W**] [*objfile* [*corfile* [*directory*]]]

DESCRIPTION

sdb is a symbolic debugger which can be used with C and **f77**(1) programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

Objfile is normally an executable program file which has been compiled with the **–g** (debug) flag option; if it has not been compiled with the **–g** flag option, or if it is not an executable file, the symbolic capabilities of **sdb** are limited, but the file can still be examined and the program debugged. The default for *objfile* is **a.out**. *Corfile* is assumed to be a core image file produced after executing *objfile*; the default for *corfile* is **core**. The core file need not be present. A **–** in place of *corfile* forces **sdb** to ignore any core image file. Source files used in constructing *objfile* must be in *directory* to be located.

It is useful to know that at any time there is a *current line* and *current file*. If *corfile* exists then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in **main()**. The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfile* cannot be found, or are newer than *objfile*. This checking feature and the accompanying warnings may be disabled by the use of the **–W** flag.

Names of variables are written just as they are in C or **f77**(1). Variables local to a procedure may be accessed using the form *procedure***:***variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable***.***member*, pointers to structure members as *variable***–>***member*, and array elements as *variable***[***number***]**. Pointers may be dereferenced by using the form *pointer***[0]**. Combinations of these forms may also be used. **f77** common variables may be referenced by using the name of the common block instead of the structure name. Blank common variables may be named by the form **.***variable*. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by **sdb**. An unqualified structure variable may also be used with various commands. Generally, **sdb** interprets a structure as a set of variables; thus, it displays the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value **sdb** displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as

variable[*number*][*number*]...,

or as

variable[*number*,*number*,...].

In place of *number*, the form *number*;*number* may be used to indicate a range of values, * may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, **sdb** displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted. A multidimensional parameter in an f77 program cannot be displayed as an array, but it is actually a pointer, whose value is the location of the array. The array itself can be accessed symbolically from the calling function.

A particular instance of a variable on the stack may be referenced by using the form *procedure*:*variable*,*number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal, or hexadecimal.

Line numbers in the source program are referred to as *file-name*:*number* or *procedure*:*number*. In either case the number is relative to the beginning of the file. If no procedure or filename is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under **sdb** all addresses refer to the executing program; otherwise they refer to *objfile* or *corfile*. An initial argument of **-w** permits overwriting locations in *objfile*.

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*). The *file address* corresponding to a written *address* is calculated as follows:

b1address<e1

file address=address+f1−b1

or

b2address<e2

file address=address+f2−b2

Otherwise, the requested *address* is not legal. In some cases (e.g., for programs with separated I and D space) the two segments for a file may overlap.

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.

In order for sdb to be used on large files, all appropriate values are kept as signed 32-bit integers.

Commands.

The commands for examining data in the program are:

t Print a stack trace of the terminated or halted program.

T Print the top line of the stack trace.

variable/clm

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

b	one byte
h	two bytes (half word)
l	four bytes (long word)

Legal values for *m* are:

c	character
d	decimal
u	decimal, unsigned
o	octal
x	hexadecimal
f	32-bit single precision floating point
g	64-bit double precision floating point
s	Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.
a	Print characters starting at the variable's address. This format may not be used with register variables.
p	pointer to procedure
i	Disassemble machine-language instruction with addresses printed numerically and symbolically.
I	Disassemble machine-language instruction with addresses printed numerically only.

The length specifiers are only effective with the formats **c**, **d**, **u**, **o** and **x**. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, sdb chooses a length and a format suitable for the variable's type, as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells sdb to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or, if no length is given, by the size associated with the *variable*. If a count specifier is used for the s or a command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command */*.

The sh(1) metacharacters * and ? may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified, only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

linenumber?lm
variable:?lm

Print the value at the address from a.out or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

variable=lm
linenumber=lm
number=lm

Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then lx is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

variable!value

Set *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions that produce more than one value, such as structures, are not allowed. Character constants are denoted *'character*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression that indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

x Print the machine registers and the current machine-language instruction.

X Print the current machine-language instruction.

The commands for examining source files are:

e *procedure*

e *file-name*
e *directory/*
e *directory file-name*

The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, filename, or directory is given, the current procedure name and filename are reported.

/regular expression/

Search forward from the current line for a line containing a string matching *regular expression* as in ed(1). The trailing / may be elided.

?regular expression?

Search backward from the current line for a line containing a string matching *regular expression* as in ed(1). The trailing ? may be elided.

p Print the current line.

z Print the current line followed by the next 9 lines. Set the current line to the last line printed.

w Window. Print the 10 lines around the current line.

number

Set the current line to the given line number. Print the new current line.

count **+**

Advance the current line by *count* lines. Print the new current line.

count **-**

Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

count **r** *args*
count **R**

Run the program with the given arguments. The **r** command with no arguments reuses the previous arguments to the program while the **R** command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber **c** *count*
linenumber **C** *count*

Continue after a breakpoint or interrupt. If *count* is given, it specifies the number of breakpoints to be ignored. **C** continues with the signal that caused the program to stop reactivated and **c** ignores it. If a linenumber is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

linenumber **g** *count*

Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

s *count*
S *count*

Single step the program through *count* lines. If no count is given then the program is run for one line. S is equivalent to s except it steps through procedure calls.

i
I Single step by one machine-language instruction. I steps with the signal that caused the program to stop reactivated and i ignores it.

*variable***$m** *count*
*address***:m** *count*

Single step (as with s) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

level **v**

Toggle verbose mode, for use when single stepping with S, s, or m. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A v turns verbose mode off if it is on for any level.

k Kill the program being debugged.

procedure(*arg1*,*arg2*,...)
procedure(*arg1*,*arg2*,...)/*m*

Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to d.

linenumber **b** *commands*

Set a breakpoint at the given line. If a procedure name without a line number is given (e.g., proc:), a breakpoint is placed at the first line in the procedure even if it was not compiled with the −g flag option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to sdb. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If **k** is used as a command to execute at a breakpoint, control returns to sdb, instead of continuing execution.

B Print a list of the currently active breakpoints.

linenumber **d**

Delete a breakpoint at the given line. If no *linenumber* is given, the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d**, the breakpoint is deleted.

D Delete all breakpoints.

l Print the last executed line.

linenumber **a**

Announce. If *linenumber* is of the form *proc*:*number*, the command effectively does a *linenumber* **b***l*. If *linenumber* is of the form *proc*:, the command effectively does a *proc*: **b T**.

Miscellaneous commands:

!*command*

The command is interpreted by sh(1).

newline

If the previous command printed a source line, advance the current line by one line and print the new current line. If the previous command displayed a memory location, display the next memory location.

CONTROL-D

Scroll. Print the next 10 lines of instructions, source, or data, depending on which was printed last.

< *filename*

Read commands from *filename* until the end of file is reached, then continue to accept commands from standard input. When sdb is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

M Print the address maps.

M[/] [*] *b e f*

Record new values for the address map. The arguments ? and / specify the text and data maps, respectively. The first segment, (*b1*, *e1*, *f1*), is changed unless * is specified, in which case the second segment, (*b2*, *e2*, *f2*), of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

" *string*

Print the given string. The C escape sequences of the form *character* are recognized, where *character* is a nonnumeric character.

q Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

V Print the version number.
Q Print a list of procedures and files being debugged.
Y Toggle debug output.

FILES

/usr/bin/sdb
a.out
core

SEE ALSO

adb(1), cc(1), ctrace(1), f77(1), sh(1), a.out(4), core(4).

WARNINGS

Data stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

BUGS

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

The default type for printing f77 parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing f77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.

The range of an f77 array subscript is assumed to be *1* to *n*, where *n* is the dimension corresponding to that subscript. This is only significant when the user omits a subscript, or uses * to indicate the full range. There is no problem in general with arrays having subscripts whose lower bounds are not 1.

NAME

sdiff − side-by-side difference program

SYNOPSIS

sdiff [options ...] **file1 file2**

DESCRIPTION

Sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a **<** in the gutter if the line only exists in *file1*, a **>** in the gutter if the line only exists in *file2*, and a | for lines that are different.

The following options exist:

−w *n* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters. The width must be between 20 and 200.

−l Only print the left side of any lines that are identical.

−s Do not print identical lines.

−o *output* Use the next argument, *output*, as the name of a third file that is created as a user controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l append the left column to the output file
r append the right column to the output file
s turn on silent mode; do not print identical lines
v turn off silent mode
e l call the editor with the left column
e r call the editor with the right column
e b call the editor with the concatenation of left and right
e call the editor with a zero length file
q exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

EXAMPLE

```
If "file1" contains:   x
                       a
                       b
                       c
                       d
and "file2" contains:  y
                       a
                       d
                       c
```

then

```
sdiff file1 file2
```

would print:

```
x   |   y
a       a
b   <
c   <
d       d
    >   c
```

SEE ALSO

diff(1), ed(1).

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** script] [**–f** sfile] [files]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **–f** option causes the script to be taken from file *sfile*. The **–e** option causes the script to be taken directly from the command line. These options accumulate, so many scripts can be used in one invocation of the command. If there is just one **–e** option and no **–f** options, the flag **–e** may be omitted. Note that all shell metacharacters must be quoted when a command line script is supplied, so care must be taken when using the **–e** option.

The **–n** option suppresses the default output; output will be generated only if explicitly requested by certain **sed** commands (**p**, **P**, **i**, and the **p** option of the **s** command).

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed* (1) modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line *embedded* in the pattern space.

A period . matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**\
text Append. Place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the **:** command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) **c**\
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) i\
text Insert. Place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) **p** Print. Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s** */regular expression /replacement /flags*

Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed* (1). *Flags* is zero or more of:

- **n** n= 1 - 512. Substitute for just the nth occurrence of the *regular expression.*
- **g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
- **p** Print the pattern space if a replacement was made.
- **w** *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a t. If *label* is empty, branch to the end of the script.

(2) **w** *wfile* Write. Append the pattern space to *wfile*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) **y** */string1 /string2 /*

Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function*

Don't. Apply the *function* (or group, if *function* is {} only to lines *not* selected by the address(es).

(0) **:** *label* This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) **=** Place the current line number on the standard output as a line.

(2) **{** Execute the following commands through a matching **}** only when the pattern space is selected.

(0) An empty command is ignored.

(0) **#** If a **#** appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the **#** is an 'n', then the default output will be suppressed, just as if the **−n** command option had been invoked. The rest of the line after **#n** is also ignored. It is an error for the **#** command to be used on any line but the first line of the file. A script file must contain at least one non-comment line.

EXAMPLE

```
sed -f sedfile inputfile > filea
```

will process the "inputfile" according to the *sedfile* script, and place the results in "filea".

The *sedfile* script

```
4 a\
XXXXXXXXXXXXX
```

would insert a row of Xs after line 4.

SEE ALSO

awk(1), ed(1), grep(1).

WARNING

Operations based on a deleted line are lost. For example, if you insert text before line 4 and then delete line 4, the inserted text is lost. Reads at line 0 are actually reads before line 1, so deleting line 1 erases these reads. Writes are lost as well, although the filename is created.

NAME

sh, rsh – shell, the standard/restricted command programming language

SYNOPSIS

sh [–acefhiknrstuvx] [args]
rsh [–acefhiknrstuvx][args]

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters **, @, #, ?, –, $,** and **!**.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by | (or, for historical compatibility, by ^). The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, **&**, **&&**, or |0, and optionally terminated by ; or **&**. Of these four symbols, ; and **&** have equal precedence, which is lower than that of **&&** and |0 . The symbols **&&** and |0 also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (|0) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**
Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation*) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)
Execute *list* in a sub-shell.

{*list*;}
list is simply executed.

name () {*list*;}
Define a function which is referenced by *name* . The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (‘ ’) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution

The character **$** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] . . .

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
: The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter* :–*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter* :=*word*}
: If *parameter* is not set or is null, set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter* :?*word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter* :+*word*}
: If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-‘pwd‘}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

– Flags supplied to the shell on invocation or by the set command.

? The decimal value returned by the last synchronously executed command.

$$ The process number of this shell.

! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.

CDPATH The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default: **$**.

PS2 Secondary prompt string, by default: **>**.

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

SHACCT If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1) and *acctcms*(1M) can be used to analyze the data collected.

SHELL When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an "r" in the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, and **IFS**. **HOME** and **MAIL** are set by *login*(1)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in IFS) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters *****, **?**, and **[**. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character **.** at the start of a file name or immediately following a **/**, as well as the character **/** itself, must be matched explicitly.

***** Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!", any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \\. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \\ quotes the characters \, ‘ , ", and **$**. “**$***” is equivalent to “**$1 $2** …, whereas “**$@**” is equivalent to “**$1**” “**$2**” ….

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[–] word The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\new-line** is ignored, and \ must be used to quote the characters \, $, ‘ , and the first character of *word*. If – is appended to <<, all leading tabs are stripped from *word* and from the document.

<&digit Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&digit**.

<&– The standard input is closed. Similarly for the standard output using **>&-**.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
    ... 1>xxx 2>&1
```

first associates file descriptor 1 with file *xxx* . It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx* .

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see *environ* (5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the values of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
    TERM=450 cmd                          and
    (export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **–k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
    echo a=b c
```

```
set –k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If a command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec* (2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands.

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file*

Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]

Exit from the enclosing **for** or **while** loop, if any. If *n* is specified, break *n* levels.

continue [*n*]

Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]

Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

echo [*arg* ...]

Echo arguments. See *echo*(1) for usage and description.

eval [*arg* ...]

The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]

The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]

Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]

The given *name* s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

hash [−r] [*0me* **...**]

For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The -r option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]

Equivalent to **exec newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]

One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]

The given *name* s are marked *readonly* and the values of the these *name* s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]

Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**—aefhkntuvx [** *arg* ...]]

−a Mark variables which are modified or created for export.

−e Exit immediately if a command exits with a non-zero exit status.

−f Disable file name generation.

−h Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).

−k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

−n Read commands but do not execute them.

−t Exit after reading and executing one command.

−u Treat unset variables as an error when substituting.

−v Print shell input lines as they are read.

−x Print commands and their arguments as they are executed.

— Do not change any of the flags; useful in setting **$1** to −.

Using + rather than − causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given, the values of all names are printed.

shift [*n*]

The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test* (1) for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [**name** ...]

For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [**−f**] [*n*]

imposes a size limit of *n*

−f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.

If no option is given, **−f** is assumed.

umask [*nnn*]

The user file-creation mask is set to *nnn* (see *umask* (2)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK,** and **IFS** cannot be unset.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through *exec* (2) and the first character of argument zero is –, commands are initially read from **/etc/profile** and from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; Note that unless the **–c** or **–s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

–c *string* If the **–c** flag is present, commands are read from *string*.

–s If the **–s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

–i If the **–i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case, TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

–r If the **–r** flag is present, the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd* (1)),
setting the value of **$PATH,**
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the exit command above).

EXAMPLE

```
sh -x script1
```

will execute each command in "script1", echoing the command just before executing it.

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

acctcom(1), cd(1), echo(1), env(1), login(1), newgrp(1), pwd(1), test(1), umask(1), acctcms(1M), dup(3), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), environ(5), profile(4).

CAVEATS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

BUGS

Filename pattern matching is not done on redirected I/O filenames.

NAME

shl − shell layer manager

SYNOPSIS

shl

DESCRIPTION

Shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty* option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *Shl* has its own prompt, **>>>**, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (**/dev/sxt/???**). The virtual device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* **(1)** through **(7)** cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create [*name*]

Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer. The shell prompt variable **PS1** is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block *name* [*name* ...]

For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

delete *name* [*name* ...]

For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)).

help (or **?**)

Print the syntax of the *shl* commands.

layers [**−l**] [*name* ...]

For each *name*, list the layer name and its process group. The **−l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **-loblk** within the layer.

quit Exit *shl*. All layers are sent the SIGHUP signal.

name Make the layer referenced by *name* the current layer.

FILES

/dev/sxt/???	Virtual tty devices
$SHELL	Variable containing path name of the shell to use (default is /bin/sh).

SEE ALSO

sh(1), stty(1). ioctl(2), signal(2), sxt(7).

NAME

size – print section sizes of common object files

SYNOPSIS

size [–d] [–o] [–V] ***filename*****(s)**

DESCRIPTION

The **size** command produces section size information for each section in the common object files. The name of the section is printed followed by its size in bytes, physical address, and virtual address.

Numbers are printed in hexadecimal unless either the **–o** or the **–d** flag option is used, in which case they are printed in octal or in decimal, respectively.

The **–V** flag supplies the version information on the command.

FILES

/bin/size

SEE ALSO

as(1), cc(1), ld(1), a.out(4).

DIAGNOSTICS

size: *name:* cannot open	*name* cannot be read.
size: *name:* bad magic	*name* is not an object file.

NAME

size − print section sizes of common object files

SYNOPSIS

size [−**o**] [−**d**] [−**V**] filename(s)

DESCRIPTION

The *size* command produces section size information for each section in the common object files. The name of the section is printed followed by its size in bytes, physical address, and virtual address.

Numbers are printed in hexadecimal unless either the −**o** or the −**d** option is used, in which case they are printed in octal or in decimal, respectively.

The −**V** flag supplies the version information on the *size* command.

SEE ALSO

as(1), cc(1), ld(1).

DIAGNOSTICS

size: name: cannot open	*Name* cannot be read.
size: name: bad magic	*Name* is not an object file.

NAME

size5.0 − size of an object file

SYNOPSIS

size5.0 [−**x**] [object ...]

DESCRIPTION

Size5.0 prints the decimal number of bytes required by the text, data, and bss portions, and their sum in decimal and (hexidecimal), of each object-file argument. If no file is specified, **a.out** is used.

If the −**x** option is specified, *size5.0* prints the hexidecimal number of bytes required by the text, data, and bss portions, and their sum in hexidecimal and (decimal).

EXAMPLE

```
        size5.0
```

prints the number of bytes for the various portions of the *a.out* file, and their sum in decimal and hexidecimal.

SEE ALSO

a.out5.0(4).

NAME

sleep — suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
      (sleep 105; command)&
```

or to execute a command every so often, as in:

```
            while true
            do
                    command
                    sleep 37
            done
```

EXAMPLE

label:

```
            command >> x
            command >> x
            date >> x
            sleep 10
            goto label
```

The preceding *sh*(1) script would execute the two commands and append the results to file "x", then sleep for 10 seconds and repeat the process.

SEE ALSO

alarm(2), sleep(3C).

BUGS

Time must be less than 65536 seconds.

NAME

sno − SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

Sno is a SNOBOL compiler and interpreter (with slight differences). *Sno* obtains input from the concatenation of the named *file*s 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 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 done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

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 spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable **sysppt** is not available.

SEE ALSO

awk(1).
SNOBOL, a String Manipulation Language, by D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* **11** (1964), pp. 21-30.

NAME

sort – sort and/or merge files

SYNOPSIS

sort [**–c**] [**–m**] [**–u**] [**–o** *output*] [**–y** *kmem*] [**–z** *recsz*] [**–d**] [**–f**] [**–i**] [**–M**] [**–n**] [**–r**] [**–b**] [**–t** *x]* [*+pos1* [*–pos2*]] [*files*]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if – is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following flag options alter the default behavior:

–c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

–m Merge only, the input files are already sorted.

–u Unique: suppress all but one in each set of lines having equal keys.

–o *output*

The argument given 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. There may be optional blanks between **–o** and *output.*

–y *kmem*

The amount of main memory used by the sort has a large impact on its performance. sorting a small file in a large amount of memory is a waste. If this flag option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this flag option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, –y0 is guaranteed to start with minimum memory. By convention, –y (with no argument) starts with maximum memory.

–z *recsz*

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **–c** or **–m** flag options, a popular system default size will be used.

Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following flag options override the default ordering rules.

–d "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

–f Fold lower case letters into upper case.

–i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

–M Compare as months. The first three nonblank characters of the field are folded to upper case and compared so that "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **–M** flag option implies the **–b** flag option (see below).

–n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **–n** flag option implies the **–b** flag option (see below). Note that the **–b** flag option is only effective when restricted sort key specifications are in effect.

–r Reverse the sense of comparisons.

When ordering flag options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering flag options override all global ordering flag options for that key.

The notation *+pos1 –pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *–pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a *newline*. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the flag options:

–b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **–b** flag option is specified before the first

+pos1 argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *−pos2* argument (see below).

−t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first nonblank in the *m*+1st field; +*m*.0b refers to the first nonblank character in the *m*+1st field.

A last position specified by *−m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; −*m*.1b refers to the first nonblank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

sort the contents of *infile* with the second field as the sort key:

```
sort +1 −2 infile
```

sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort −r −o outfile +1.0 −1.2 infile1 infile2
```

sort, in reverse order, the contents of *infile1* and *infile2* using the first nonblank character of the second field as the sort key:

```
sort −r +1.0b −1.1b infile1 infile2
```

Print the password file (**passwd**(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort −t: +2n −3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the flag options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort –um +2 –3 infile
```

FILES

/bin/sort
/usr/tmp/stm???

SEE ALSO

comm(1), join(1), rev(1), sortbib(1), tsort(1), Buniq(1).

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the –c flag option.

When the last line of an input file is missing a newline character, *sort* appends one, prints a warning message, and continues.

NAME

spell, hashmake, spellin, hashcheck − find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−x**] [**−l**] [**−i**] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **−b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(1)) follows chains of included files (**.so** and **.nx** *troff*(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **−l** option, *spell* will follow the chains of *all* included files. Under the **−i** option, *spell* will ignore all chains of included files.

Under the **+***local_file* option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy−y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin n Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

EXAMPLE

```
spell filea fileb filec > misteaks
```

would put a list of the words from "filea", "fileb" and "filec" that were not part of the on-line dictionary into file "misteaks".

The following example creates the hashed spell list **hlist** and checks the result by comparing the two temporary files; they should be equal.

```
cat goodwds | /usr/lib/spell/hashmake | sort −u >tmp1
cat tmp1 | /usr/lib/spell/spellin ‘cat tmp1 | wc −l‘ >hlist
cat hlist | /usr/lib/spell/hashcheck >tmp2
diff tmp1 tmp2
```

The on-line dictionary rejects technical terms and proper names it does not know and treats them as misspellings.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1).

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.
The British spelling feature was done by an American.

NAME

spline − interpolate smooth curve

SYNOPSIS

spline [options]

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., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted.

The following *options* are recognized, each as a separate argument:

−a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

−k The constant k used in the boundary value computation: $y_0'' = ky_1''$, $y_n'' = ky_{n-1}''$

is set by the next argument (default $k = 0$).

−n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).

−p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.

−x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

EXAMPLE

```
spline -n 10 > spline.out
0 0
1 2
2 4
3 9
```

will create the file "spline.out" with the contents:

```
3.000000  8.999999
2.666667  7.096296
2.333333  5.370370
2.000000  4.000000
1.666667  3.096296
1.333333  2.503703
1.000000  2.000000
0.666667  1.407407
0.333333  0.725926
0.000000  0.000000
```

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 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 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. 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.

EXAMPLE

```
split -100 filea newfile
```

would split "filea" into 100-line pieces and put them in "newfileaa", "newfileab", and so forth until the end of filea.

SEE ALSO

bfs(1), csplit(1).

NAME

ssp – make output single spaced

SYNOPSIS

ssp [name ...]

DESCRIPTION

Ssp removes extra blank lines, compressing two or more blank lines into one. It can be used directly, or as a filter after *nroff* or other text formatting operations.

The – option removes all blank lines.

EXAMPLE

```
nroff -ms filea fileb | ssp >> filec
```

would *nroff* the files with the –ms macro package, then single space the output and direct it to "filec".

NAME

strings − find the printable strings in an object, or other binary file

SYNOPSIS

strings [−] [−o] [−*number*] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. Unless the − flag is given, *strings* only looks in the initialized data space of object files. If the **−o** flag is given, then each string is preceded by its offset in the file (in octal). If the −*number* flag is given, then number is used as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

EXAMPLE

```
strings obj1
```

will locate the ASCII-character strings in the object file "obj1".

SEE ALSO

od(1).

BUGS

The algorithm for identifying strings is extremely primitive.

NAME

strip − strip symbol and line number information from an object file

SYNOPSIS

strip [−l] [−x] [−r] [−s] [−V] filename(s)

DESCRIPTION

The *strip* command strips the symbol table and line number information from object files, including archives. When *strip* has been performed, no symbolic debugging access is available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using the following options:

−l Strip line number information only; do not strip any symbol table information.

−x Do not strip static or external symbol information.

−r Reset the relocation indexes into the symbol table.

−s Reset the line number indexes into the symbol table (do not remove). Reset the relocation indexes into the symbol table.

−V Print the version of the *strip* command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* complains and terminates without stripping *filename* unless the **−r** flag is used.

If the *strip* command is executed on a common archive file (see *ar*(4)) the archive symbol table is removed. The archive symbol table must be restored by executing the *ar*(1) command with the s option before the archive can be link edited by the *ld*(1) command. *Strip*(1) instructs the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

FILES

/usr/tmp/str??????

SEE ALSO

as(1), cc(1), ld(1), ar(4), a.out(4).

DIAGNOSTICS

strip: name: cannot open	*name* cannot be read.
strip: name: bad magic	*name* is not an object file.
strip: name: relocation entries present; cannot strip	*name* contains relocation entries and the **−r** flag was not used; therefore, the symbol table information cannot be stripped.

NAME

strip – strip symbol and line number information from an object file

SYNOPSIS

strip [–l] [–r] [–s] [–V] [–x] *files*

DESCRIPTION

The **strip** command strips the symbol table and line number information from object files, including archives. When **strip** has been performed, no symbolic debugging access is available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using the following flag options:

–l Strip line number information only; do not strip any symbol table information.

–x Do not strip static or external symbol information.

–r Reset the relocation indexes into the symbol table.

–s Reset the line number indexes into the symbol table (do not remove). Reset the relocation indexes into the symbol table.

–V Print the version of the **strip** command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, **strip** complains and terminates without stripping *filename* unless the **–r** flag is used.

If the **strip** command is executed on a common archive file (see **ar**(4)) the archive symbol table is removed. The archive symbol table must be restored by executing the **ar**(1) command with the **s** flag option before the archive can be link edited by the **ld**(1) command. **strip**(1) instructs the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

FILES

/bin/strip
/usr/tmp/str??????

SEE ALSO

as(1), **cc**(1), **ld**(1), **sdb**(1), **ar**(4), **a.out**(4).

DIAGNOSTICS

strip: ***name:*** **cannot open**

name cannot be read.

strip: ***name:*** **bad magic**

name is not an object file.

strip: ***name*****: relocation entries present; cannot strip**

name contains relocation entries and the −r flag was not used; therefore, the symbol table information cannot be stripped.

NAME

strip5.0 − remove symbols and relocation bits

SYNOPSIS

strip5.0 name ...

DESCRIPTION

Strip5.0 removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. This is useful to save space after a program has been debugged.

The effect of *strip5.0* is the same as use of the −s option of *ld*.

If *name* is an archive file, *strip5.0* will remove the local symbols from any *a.out5.0* format files it finds in the archive. Certain libraries, such as those residing in **/lib**, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

EXAMPLE

```
strip5.0 a.out5.0
```

removes the symbol table and relocation bits from **a.out5.0**.

FILES

/tmp/stm*	temporary file

SEE ALSO

ld5.0(1).

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**−n**] [filename] [**−a**] [**−g**] [options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input. Without arguments, it reports the settings of certain options.

With the **−n filename** option, *stty* will open the file specified by *filename* with the option O_NODELAY (this means that it will open modem-controlled lines immediately instead of waiting for a carrier) and uses it as standard input.

With the **−a** option, it reports all of the option settings.

With the **−g** option, it reports current settings in a form that can be used as an argument to another *stty* command.

Detailed information about the modes listed in the first five groups below may be found in *termio*(7) in the *Administrator Manual*. Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (−parenb) enable (disable) parity generation and detection.

parodd (−parodd) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *termio*(7)).

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. (All speeds are not supported by all hardware interfaces.) **19200** is equivalent to **exta**. **38400** is equivalent to **extb**.

hupcl (−hupcl) hang up (do not hang up) DATA-PHONE connection on last close.

hup (−hup) same as **hupcl (−hupcl)**.

cstopb (−cstopb) use two (one) stop bits per character.

cread (−cread) enable (disable) the receiver.

clocal (−clocal) assume a line without (with) modem control.

loblk (−loblk) block (do not block) output from a non-current layer.

Input Modes

ignbrk (−ignbrk) ignore (do not ignore) break on input.

brkint (−brkint) signal (do not signal) INTR on break.

ignpar (−ignpar) ignore (do not ignore) parity errors.

parmrk (−parmrk) mark (do not mark) parity errors (see *termio*(7)).

inpck (−inpck) enable (disable) input parity checking.

istrip (−istrip) strip (do not strip) input characters to seven bits.

inlcr (−inlcr) map (do not map) NL to CR on input.

igncr (−igncr) ignore (do not ignore) CR on input.

icrnl (−icrnl) map (do not map) CR to NL on input.

iuclc (−iuclc) map (do not map) upper-case alphabetics to lower case on input.

ixon (−ixon) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (−ixany) allow any character (only DC1) to restart output.

ixoff (−ixoff) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (−opost) post-process output (do not post-process output; ignore all other output modes).

olcuc (−olcuc) map (do not map) lower-case alphabetics to upper case on output.

onlcr (−onlcr) map (do not map) NL to CR-NL on output.

ocrnl (−ocrnl) map (do not map) CR to NL on output.

onocr (−onocr) do not (do) output CRs at column zero.

onlret (−onlret) on the terminal NL performs (does not perform) the CR function.

ofill (−ofill) use fill characters (use timing) for delays.

ofdel (−ofdel) fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 select style of delay for carriage returns (see *termio*(7)).

nl0 nl1 select style of delay for line-feeds (see *termio*(7)).

tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *termio*(7)).

bs0 bs1 select style of delay for backspaces (see *termio*(7)).

ff0 ff1 select style of delay for form-feeds (see *termio*(7)).

vt0 vt1 select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (–isig) enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.

icanon (–icanon) enable (disable) canonical input (ERASE and KILL processing).

xcase (–xcase) canonical (unprocessed) upper/lower-case presentation.

echo (–echo) echo back (do not echo back) every character typed.

echoe (–echoe) echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (–echok) echo (do not echo) NL after KILL character.

lfkc (–lfkc) the same as **echok (–echok)**; obsolete.

echonl (–echonl) echo (do not echo) NL.

noflsh (–noflsh) disable (enable) flush after INTR, QUIT, or SWTCH.

stwrap (–stwrap) disable (enable) truncation of lines longer than 79 characters on a synchronous line.

stflush (–stflush) enable (disable) flush on a synchronous line after every *write*(3).

stappl (–stappl) use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **eof**, **eol**, **ctab**, **min**, or **time** (**ctab** is used with **–stappl**, (**min** and **time** are used with **–icanon**; see *termio*(7)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "**^d**" is a **CTRL-d**); "**^?**" is interpreted as DEL and "**^–**" is interpreted as undefined.

line *i* set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or parity enable **parenb** and **cs7**.

oddp enable **parenb**, **cs7**, and **parodd**.

–parity, –evenp, or –oddp
disable **parenb**, and set **cs8**.

raw (−raw or cooked) enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

nl (−nl) unset (set) **icrnl**, **onlcr**. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (−lcase) set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (−LCASE) same as **lcase (−lcase)**.

tabs (−tabs or tab3) preserve (expand to spaces) tabs when printing.

ek reset ERASE and KILL characters back to normal # and @.

sane resets all modes to some reasonable values.

term set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1), ioctl(2), termio(7).

NAME

su − become super-user or another user

SYNOPSIS

su [−] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or **/bin/sh** if none is specified (see *sh*(1)). To restore normal user ID privileges, type an EOF (*cntrl-d*) to the new shell.

Any additional arguments given on the command line are passed to program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **−c** *string* executes *string* via the shell and an arg of **−r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a −, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is −, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **−sh** or **−su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of −*program* by both *login*(1) and *su*(1).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

EXAMPLE

```
su unisoft
```

would cause the system to prompt for UniSoft's password; if the password is typed in correctly, UniSoft's identity is substituted for yours, so far as the system is concerned.

To become user **bin** while retaining your previously exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permission of user **bin**, type:

```
        su − bin −c "command args"
```

FILES

/etc/passwd	system's password file
/etc/profile	system's profile
$HOME.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1), passwd(4), profile(4), environ(5).

NAME

sum — print checksum and block count of a file

SYNOPSIS

sum [−r] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **−r** causes an alternate algorithm to be used in computing the checksum.

EXAMPLE

```
sum filea
```

produces the checksum and the block count of "filea".

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

sum7 − sum and count blocks in a file

SYNOPSIS

sum7 file

DESCRIPTION

Sum7 calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file, to the nearest whole block. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

EXAMPLE

```
sum7 sum7.1
```

produces the checksum and the block count of this manual section, namely:

```
21009   1
```

SEE ALSO

wc(1).

NAME

sumdir — sum and count characters in the files in the given directories

SYNOPSIS

sumdir [directories]

DESCRIPTION

Sumdir calculates and prints a 16-bit checksum for the named file, and also prints the number of characters in the file. It is typically used to look for bad spots on the file system, or to validate a file transmitted over some transmission line. The output from this program differs from the output from the *sum*(1) program in that *sumdir* prints the number of characters rather than the number of blocks in the file.

Sumdir provides a recursive checksum of all files in the specified directory.

EXAMPLE

```
sumdir man1
```

produces the checksum and the character count of the files in the directory **man1**.

SEE ALSO

sum(1).

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. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

EXAMPLE

```
sync
```

should be typed to flush all internal disk buffers, before bringing down the system.

SEE ALSO

sync(2).

NAME

tabs − set tabs on a terminal

SYNOPSIS

tabs [tabspec] [**+m** n] [**−T** type]

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 have remotely-settable hardware tabs.

Users of GE 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 on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned", repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **−8**, i.e., UNIX "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the left-most column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

−*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. Files using this tab setup should include a format specification as follows:
<:t−c2 m6 s66 d:>

−c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more 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* **−h** option for high-speed output. Another special case is the value **−0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the 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. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **−8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(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:

−T*type* *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(5). If no −T flag is supplied, *tabs* searches for the **$TERM** value in the *environment* (see *environ*(5)). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (left-most) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLE

```
tabs -c
```

will send commands to the terminal to remotely set the tabs for COBOL format.

```
tabs 6,12,18
```

will set tabs in columns 6, 12 and 18.

```
tabs -10
```

will set tabs in columns 11, 21, 31, 41, 51, 61, and 71.

DIAGNOSTICS

illegal tabs when arbitrary tabs are ordered incorrectly.

illegal increment when a zero or missing increment 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.

SEE ALSO

nroff(1), pr(1), environ(5), term(5), tset(1).

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.
It is generally impossible to usefully change the left margin without also setting tabs.
Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [±[number] [**lbc** [**f**]]] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *–number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the *–f* ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process.

EXAMPLE

```
tail –f fred
```

will print the last ten lines of the file "fred", followed by any lines that are appended to "fred" between the time *tail* is initiated and killed.

```
tail –15cf fred
```

will print the last 15 characters of the file "fred", followed by any lines that are appended to "fred" between the time *tail* is initiated and killed.

SEE ALSO

cat(1), head(1), more(1), pg(1).

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

take — takes a file from a remote machine

SYNOPSIS

take [**−p** *port*] [**−s**SPEED] [**−i** [ID]] *fromfile* [*tofile*]
take [**−p** *port*] [**−s**SPEED] **−c** *command* [*args*] ...

DESCRIPTION

Take is part of system of programs useful for transferring files between UNIX systems. It is the "downloader" designed to transmit files from a remote machine to a local machine. For a brief discussion of the take/put system and installation instructions, see **TAKE/PUT** in the *Administrator Guide*.

Take transfers a file, directory, or output from a command given at a remote machine. The default *port* is **/dev/tty0**; the **−p** option can be used to specify an alternate port. The default *speed* is determined by the system; the **−s** option can be used to specify a specific speed. The **−i** [ID] option remaps pathnames on the remote machine. The ID (if present) is passed to the remote machine where it is used to locate a line containing pathname prefixes (using the **/etc/takelist** file discussed below). If no ID is given after the **−i** flag, then the default system ID is read from the **/etc/sys_id** file (if it exists); otherwise **take7** will use the account name of the invoker (i.e., the person who logged in to the port used) to determine which line of **/etc/takelist** to apply. See the overview document for details of the mapping.

The **−c** option is useful for executing an arbitrary command on the remote machine. All arguments following the **−c** flag are collected, transmitted to the remote machine and executed as a single command. The standard output and standard error from this command are returned as the standard output and standard error of **take**.

In order to perform its function, *take*(1C) interfaces with the program **/usr/bin/take7** on the remote machine.

EXAMPLE

```
take /a/b/c
```

takes the contents of the directory (or file) "/a/b/c" on the remote machine and copies them into a similarly named directory (or file) on the local machine; if "/a/b/c" did not previously exist on the local machine, it is created; otherwise it is overwritten.

```
take file.c /x/y/z/filename
```

takes the contents of "file.c" from the remote machine and copies them into "/x/y/z/filename" on the local machine. Note that if "filename" exists on the local machine, its contents are overwritten.

FILES

fromfile The remote file name. When using the **−i** option, this file should usually be specified as a pathname starting at the root of the local machine.

tofile The local file name; if *tofile* is null, *tofile* is defaulted to *fromfile*. If *tofile* is a directory, then *tofile* has the last segment of the *fromfile* path appended to it.

SEE ALSO

cu(1C), put(1C)
TAKE/PUT in the *Administrator Guide*.

NAME

talk — talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

Talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on you own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
user@host

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, it sends the message

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

```
talk  your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as his login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control L will cause the screen to be reprinted, while your erase, and kill characters will work in talk as normal. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg* command. At the outset talking is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

FILES

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

SEE ALSO

mesg(1), who(1), mail(1), write(1)

NAME

tar − tape file archiver

SYNOPSIS

tar [key] [files]

DESCRIPTION

Tar saves and restores files on magnetic tape. 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 *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named *files* are written on the end of the tape. This may not work on all media. It requires the ability to "seek." The **c** function implies this function.

x The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. If a named file on tape does not exist on the system, the file is created with the same mode as the one on tape except that the set-user-ID and set-group-ID bits are not set unless you are super-user. If the files exist, their modes are not changed except for the bits described above. The owner, group, and modification time are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t The names of all the files on the tape are listed.

u The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape.

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This command implies the **r** function.

The following characters may be used in addition to the letter that selects the desired function:

#s Where **#** is a tape drive number (**0**,...,**7**), and s is the density (**l** - low (800 bpi), **m** - medium (1600 bpi), or **h** - high (6250 bpi)). This modifier selects the drive on which the tape is mounted. The default is **0m**.

v Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w Causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means "no".

f causes *tar* to use the next argument as the name of the archive instead of **/dev/mt/??**. If the name of the file is −, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf − . | (cd todir; tar xf −)
```

b Causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **x** and **t**).

l Tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m tells *tar* not to restore the modification times. The modification time of the file will be the time of extraction.

o Causes extracted files to take on the user and group identifier of the user running the program rather than those on the tape.

This version of *tar* is capable of writing more than one tape or disk. The user will be prompted to change media when necessary. The next two options are used for tapes; the last is for disks.

d causes *tar* to use the next argument as the tape's density. The default density is 1600BPI.

s causes *tar* to use the next argument as the tape's length in feet. The default length is 2300 feet.

B causes *tar* to use the next argument as the number of 512-byte blocks in the disk.

The *tar* header format is as follows:

```
# define  TBLOCK  512
# define  NBLOCK  40
# define  NAMSIZ  100
union hblock {
          char dummy[TBLOCK];
          struct header {
                    char name[NAMESIZ];
                    char mode[8];
                    char uid[8];
                    char gid[8];
                    char size[12];
                    char mtime[12];
                    char chksum[8];
                    char linkflag;
                    char linkname[NAMESIZ];
          } dbuf;
} dblock, tbuf[NBLOCK];
```

EXAMPLE

```
cd fromdir; tar cf − . | (cd todir; tar xf −)
```

will copy directories from one directory tree to another.

FILES

/dev/rmt?
/dev/mt/*
/tmp/tar*

/bin/mkdir	build directories during recovery
/bin/pwd	get working directory name

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on file-name length is 100 characters.
Empty directories are skipped when creating a tar archive.
Note that **tar c0m** is not the same as **tar cm0**.

NAME

tbl − format tables for nroff or troff

SYNOPSIS

tbl [−TX] [files]

DESCRIPTION

Tbl is a preprocessor that formats tables for *nroff* or *troff.* 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 are re-formatted by *tbl.* (The .TS and .TE command lines are not altered by *tbl*).

.TS is followed by global options. The available global options are:

center center the table (default is left-adjust);
expand make the table as wide as the current line length;
box enclose the table in a box;
doublebox enclose the table in a double box;
allbox enclose each item of the table in a box;
tab (*x*) use the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi-colon (;).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc. The available key-letters are:

c center item within the column;
r right-adjust item within the column;
l left-adjust item within the column;
n numerically adjust item in the column: units positions of numbers are aligned vertically;
s span previous item on the left into this column;
a center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
^ span down previous entry in this column;
_ replace this entry with a horizontal line;
= replace this entry with a double horizontal line.

The characters **B** and **I** stand for the bold and italic fonts, respectively; the character | indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by .TE. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ or =, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only _ or =, then that item is replaced by a single or double line.

Full details of all these and other features of *tbl* are given in the reference manual cited below.

The **−TX** option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments (or if − is specified as the last argument), *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn*(1) or *neqn*, *tbl* should come first to minimize the volume of data passed through pipes.

EXAMPLE

In the following input, ^I represents a tab (which should be typed as a genuine tab):

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population

Town^IHouseholds
^INumber^ISize
=
Bedminster^I789^I3.26
Bernards Twp.^I3087^I3.74
Bernardsville^I2018^I3.30
Bound Brook^I3425^I3.04
Bridgewater^I7897^I3.81
Far Hills^I240^I3.19
.TE
```

yields:

Household Population		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

cw(1), eqn(1), mm(1), mmt(1), nroff(1), troff(1), mm(5), mv(5)
TBL in the *Document Processing Guide*.

BUGS

See *BUGS* under *nroff*(1).

NAME

tc − phototypesetter simulator

SYNOPSIS

tc [−t] [−sn] [−pl } [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Wang Laboratories, Inc. C/A/T phototypesetter. The standard output of *tc* is intended for a Tektronix 4014 terminal with ASCII and APL character sets. The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, with overstruck combinations where necessary. Typical usage is:

troff −t files | tc

At the end of each page, *tc* waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will *suppress* the screen erase before the next page; **s** *n* will cause the next *n* pages to be skipped; and **!***cmd* will send *cmd* to the shell.

The command line options are:

−t Don't wait between pages (for directing output into a file).

−s*n* Skip the first *n* pages.

−p*l* Set page length to *l*; *l* may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

SEE ALSO

4014(1), sh(1), tplot(1G), troff(1).

BUGS

Font distinctions are lost.

NAME

tee − pipe fitting

SYNOPSIS

tee [**−i**] [**−a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The **−i** option ignores interrupts; the **−a** option causes the output to be appended to the *files* rather than overwriting them.

EXAMPLE

```
make | tee x
```

will cause the output of the make program to be recorded on file "x" as well as printed on standard output.

NAME

telnet − user interface to the TELNET protocol

SYNOPSIS

telnet [host[port]]

DESCRIPTION

Telnet is used to communicate with another host using the TELNET protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt (telnet>). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an *open* command (see below) with those arguments.

Once a connection has been opened, *telnet* enters input mode. In this mode, text typed is sent to the remote host. To issue *telnet* commands when in input mode, precede them with the *telnet* escape character (initially ^]). When in command mode, the normal terminal editing conventions are available.

The following commands are available. Only enough of each command to uniquely identify it need be typed.

open *host* **[** *port* **]**
Open a connection to the named host. If the no port number is specified, *telnet* will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*(4N)) or an Internet address specified in the dot notation.

close Close a TELNET session and return to command mode.

quit Close any open TELNET session and exit *telnet*.

z Suspend *telnet*.

escape [*escape-char*]
Set the *telnet* escape character. Control characters may be specified as ^ followed by a single letter; e.g. control-X is ^X.

status Show the current status of *telnet*. This includes the peer one is connected to, as well as the state of debugging.

options
Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations will be displayed. Options sent by *telnet* are displayed as SENT, while options received from the TELNET server are displayed as RCVD.

crmod Toggle carriage return mode. When this mode is enabled any carriage return characters received from the remote host will be mapped into a carriage return and a line feed. This mode does not affect those characters typed by the user, only those received. This mode is not very useful, but is required for some hosts that like to ask the user to do local echoing.

? [*command* **]**
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information available about the command only.

BUGS

This implementation is very simple because *rlogin*(1N) is the standard mechanism used to communicate locally with hosts.

NAME

test − condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The super-user is always granted execute permission even though 1. execute permission is meaningful only for directories and regular files, and 2. exec requires that at least one execute mode bit be set for a regular file to be executable. The following primitives are used to construct *expr*:

−r *file* true if *file* exists and is readable.

−w *file* true if *file* exists and is writable.

−x *file* true if *file* exists and is executable.

−f *file* true if *file* exists and is a regular file.

−d *file* true if *file* exists and is a directory.

−c *file* true if *file* exists and is a character special file.

−b *file* true if *file* exists and is a block special file.

−p *file* true if *file* exists and is a named pipe (fifo).

−u *file* true if *file* exists and its set-user-ID bit is set.

−g *file* true if *file* exists and its set-group-ID bit is set.

−k *file* true if *file* exists and its sticky bit is set.

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

−t [*fildes*] true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

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

−n *s1* true if the length of the string *s1* is non-zero.

s1 **=** *s2* true if strings *s1* and *s2* are identical.

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

s1 true if *s1* is *not* the null string.

n1 **−eq** *n2* true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

! unary negation operator.

−a binary *and* operator.

−o binary *or* operator (**−a** has higher precedence than **−o**).

(expr **)** parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

EXAMPLE

Test is typically used in shell scripts (*sh*(1)), as in the following example which prints the message "foo is a directory" if it is found to be one when *test*ed.

```
if test -d foo
then
        echo "foo is a dir"
fi
```

SEE ALSO

find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses [], rather than the word *test*), the square brackets must be delimited by blanks.
Some UNIX systems do not recognize the second form of the command.

Note *test* is built into **/bin/sh** and will not work in **/bin/csh**.

NAME

time — time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

EXAMPLE

```
time nroff man filea
```

will, in *sh*, perform the formatting and report the time at the end of the file, e.g.:

```
real  22.0
user   8.6
sys    6.4
```

In *csh*, on the other hand, the time report might be:

```
8.9u  7.0s  0:29  54%
```

which reports the user time, system time, real time, and percentage of real time that the CPU was active, which is the sum of the user and system times divided by real elapsed time.

SEE ALSO

timex(1), times(2).

NAME

timex − time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

Options are:

−p List process accounting records for *command* and all its children. Suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.

−o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

−s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command.* All the data items listed in *sar*(1) are reported.

EXAMPLE

```
timex ps -el
```

runs the *ps* command (with the correct options), then produces statistics concerning the command and system activity during the command to the standard error.

FILES

/usr/lib/sa/timex

SEE ALSO

acctcom(1), sar(1).

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

NAME

touch − update access and modification times of a file

SYNOPSIS

touch [**−amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used. The **−a** and **−m** options cause touch to update only the access or modification times respectively (default is **−am**). The **−c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

Note that you can't *touch* a numeric filename without preceding that filename with the date or with a non-numeric filename on the command line. For example,

```
touch 100
```

will not work, however

```
touch 0723093584 100
```

or

```
touch file1 100
```

will work.

EXAMPLE

```
touch filea fileb
```

sets the "date last modified" of the two files to the current date.

SEE ALSO

date(1), utime(2).

NAME

tp − manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or other magnetic tape. 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.

Tp is useful for importing tapes made on older systems.

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/sbo/abc** even if **/usr/sbo** 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 magnetic tapes.

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 magnetic tape as opposed to DECtape.

0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, **x** is default; for magnetic tape **0** is the default.

v Normally *tp* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

c Means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u**. This option is assumed with magnetic tape since it is impossible to selectively overwrite magnetic tape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option is known to work only with **x**.

w Causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means "yes", so the file is treated. Null response means "no", and the file does not take part in whatever is being done. Response **x** means "exit"; the *tp* command terminates immediately. In the **x** function, files previously asked about have been extracted already. With **r**, **u**, and **d**, no change has been made to the tape.

EXAMPLE

```
tp x file1
```

extracts "file1" from a *tp* formatted magnetic tape mounted on drive 0.

FILES

/dev/tap?
/dev/mt?

SEE ALSO

ar(1), cpio(1), tar(1).

DIAGNOSTICS

Several; the non-obvious one is "Phase error", which means the file changed after it was selected for dumping but before it was dumped.

BUGS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by *tp* difficult to carry to other machines; *tar*(1) avoids the problem.

Tp does not copy zero-length files to tape.

NAME

tplot − graphics filters

SYNOPSIS

tplot [**−T** terminal [**−e** raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(4)) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** (see *environ*(5)) is used. Known *terminal*s are:

300	DASI 300.
300S	DASI 300s.
450	DASI 450.
4014	Tektronix 4014.
ver	Versatec D1200A. This version of *plot* places a scan-converted image in **/usr/tmp/raster$$** and sends the result directly to the plotter device, rather than to the standard output. The **−e** option causes a previously scan-converted file *raster* to be sent to the plotter.

EXAMPLE

```
tplot -T4014 graph.out
```

will use the encoded information in "graph.out" to plot a graph on a Tektronix 4014-type terminal.

FILES

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO

plot(3X), plot(4), term(5).

NAME

tput – query terminfo database

SYNOPSIS

tput [-Ttype] capname

DESCRIPTION

Tput uses the *terminfo(4)* database to make terminal-dependent capabilities and information available to the shell. *Tput* outputs a string if the attribute (capability name) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, tput simply sets the exit code (0 for TRUE, 1 for FALSE), and does no output.

–T*type* indicates the type of terminal. Normally this flag is unnecessary, as the default is taken from the environment variable **$TERM.**

Capname indicates the attribute from the *terminfo* database. See *terminfo(4)*.

EXAMPLES

tput clear Echo clear-screen sequence for the current terminal.

tput cols Print the number of columns for the current terminal.

tput -T450 cols Print the number of columns for the 450 terminal.

bold=‘tput smso‘ Set shell variable "bold" to stand-out mode sequence for current terminal. This might be followed by a prompt:

echo "${bold}Please type in your name: \c"

tput hc Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/usr/lib/terminfo/?/*	Terminal descriptor files
/usr/include/term.h	Definition files
/usr/include/curses.h	

DIAGNOSTICS

Tput prints error messages and returns the following error codes on error:

-1	Usage error.
-2	Bad terminal type.
-3	Bad capname.

In addition, if a capname is requested for a terminal that has no value for that capname (e.g., **tput -T450 lines**), **-1** is printed.

SEE ALSO

stty(1), terminfo(4).

NAME

tr − translate characters

SYNOPSIS

tr [−cds] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. 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−z] Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a∗*n*] Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used 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.

EXAMPLE

```
tr −cs "[A−Z][a−z]" "[\012∗]" <file1 >file2
```

creates a list of all the words in "file1" one per line in "file2", where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

In this case, *tr* has substituted the *newline* character for all the alphabetics in "file1", reconstituted the alphabetics with the *−c* option, squeezed the newlines to one per occurrence, with the *−s* option, and directed the output to "file2".

SEE ALSO

ed(1), sh(1), ascii(5).

BUGS

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

troff − typeset text

SYNOPSIS

troff [options] [files]

DESCRIPTION

Troff formats text contained in *files* (standard input by default) for a Wang Laboratories, Inc., C/A/T phototypesetter. Its capabilities are described in the **NROFF/TROFF** and **TROFF Tutorial** chapters of the *Document Processing Guide* cited below.

An argument consisting of a minus (−) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

−o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end. (See *BUGS* below.)

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

−r*aN* Set register *a* (which must have a one-character name) to *N*.

−i Read standard input after *files* are exhausted.

−q Invoke the simultaneous input-output mode of the **.rd** request.

−z Print only messages generated by **.tm** (terminal message) requests.

−m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

−c*name* Prepend to the input *files* the compacted macro files
/usr/lib/macros/cmp.[nt].[dt].*name* and
/usr/lib/macros/ucmp.[nt].*name*.

−k*name* Compact the macros used in this invocation of *troff*, placing the output in files **[dt].***name* in the current directory.

−t Direct output to the standard output instead of the phototypesetter.

−f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

−w Wait until phototypesetter is available, if it is currently busy.

−b Report whether the phototypesetter is busy or available. No text processing is done.

−a Send a printable ASCII approximation of the results to the standard output.

−p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

−T*name* Use font-width tables for device *name* (the font tables are found in **/usr/lib/font/***name***/∗**). Currently, no *name*s are supported.

EXAMPLE

```
troff -o4,8-10 -mabc file1 file2
```

requests formatting of pages 4, 8, 9, and 10 of a document contained in the files named "file1" and "file2", and invokes the macro package **abc**.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/font/*	font width tables for *troff*

SEE ALSO

cw(1), eqn(1), mmt(1), nroff(1), tbl(1), tc(1), mm(5), mv(5).
NROFF/TROFF and **TROFF Tutorial** chapter of the *Document Processing Guide.*

BUGS

Troff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *troff* generates may be off by one day from your idea of what the date is.
When *troff* is used with the **−o***list* option inside a pipeline (e.g., with one or more of *cw*(1), *eqn*(1), and *tbl*(1)), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1).

EXAMPLE

```
while true
do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tset, reset − set or reset the teletype bits to a sensible state

SYNOPSIS

tset [options]
reset

DESCRIPTION

Tset causes terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. It first determines the *type* of terminal involved, names for which are specified by the **/etc/termcap** data base, and then does necessary initializations and mode settings. In the case where no argument types are specified, *tset* simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual concerns itself with type initialization, done typically once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script ".profile" (for *sh*(1) users) or ".login" (for *csh*(1) users), it is desirable to give information about the types of terminal usually used, for terminals which are connected to the computer through a modem. These ports are initially identified as being *dialup* or *plugboard* or *arpanet* etc. To specify what terminal type is usually used on these ports, **−m** is followed by the appropriate port type identifier, an optional baud-rate specification, and the terminal type to be used if the mapping conditions are satisfied. If more than one mapping is specified, the first applicable mapping prevails. A missing type identifier matches all identifiers.

Baud rates are specified as with *stty*(1), and are compared with the speed of the diagnostic output (which is almost always the control terminal). The baud rate test may be any combination of: **>**, **=**, **<**, **@**, and **!**; **@** is a synonym for = and ! inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **−m** within ′′ characters; users of *csh*(1) must also put a "\" before any "!" used here.

Thus

```
tset −m ′dialup>300:adm3a′ −m dialup:dw2 −m ′plugboard:?adm3a′
```

causes the terminal type to be set to an *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a *dw2* if the port is (otherwise) a dialup (i.e., at 300 baud or less). If the *type* above begins with a question mark, the user is asked if s/he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Thus, in this case, the user will be queried on a plugboard port as to whether they are using an *adm3a*. For other ports the port type will be taken from the **/etc/ttytype** file or a final, default *type* option may be given on the command line not preceded by a **−m**.

It is often desirable to return the terminal type, as specified by the **−m** options, and information about the terminal to a shell's environment. This can be done using the **−s** option; using the Bourne shell, *sh*(1):

```
eval `tset −s options ... `
```

or using the C shell, *csh*(1):

```
tset -s options ... > tset$$
source tset$$
rm tset$$
```

These commands cause *tset* to generate as output a sequence of shell commands which place the variables TERM and TERMCAP in the environment; see *environ*(5).

Once the terminal type is known, *tset* engages in terminal mode setting. This normally involves sending an initialization sequence to the terminal and setting the single character erase (and optionally the line-kill (full line erase)) characters.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ("#" on standard systems), the erase character is changed to a Control-H (backspace).

Other options are:

−e set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, usually ^H.

−k is similar to **−e** but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons); ^U is the preferred setting. No kill processing is done if **−k** is not specified.

−I suppresses outputting terminal initialization strings.

−Q suppresses printing the "Erase set to" and "Kill set to" messages.

−S Outputs the strings to be assigned to TERM and TERMCAP in the environment rather than commands for a shell.

Reset sets the terminal to cooked mode, turns off "cbreak" and "raw" modes, turns on "nl", and restores special characters that are undefined to their default values.

This is most useful after a program dies leaving a terminal in a funny state; you have to type **<LF>reset<LF>** to get it to work as **<CR>** often doesn't work; often none of this will echo.

It isn't a bad idea to follow *reset* with *tset*.

EXAMPLE

A typical *csh* ".login" file using *tset* would be:

```
set noglob
set term = (‘tset -e -S -r -d\?h19‘)
setenv TERM "$term[1]"
setenv TERMCAP "$term[2]"
unset term noglob
```

This ".login" sets the environment variables TERM and TERMCAP for the user's current terminal according to the file **/etc/ttytype**. If the terminal line is a dialup line, the user is prompted for the proper terminal type.

```
reset
```

returns the user's terminal to a usable state after being accidentally set by an interrupted process.

FILES

/etc/ttytype terminal id to type map database
/etc/termcap terminal capability database

SEE ALSO

csh(1), sh(1), stty(1), environ(5), ttytype(4), termcap(5).

BUGS

Should be merged with *stty*(1).

Reset doesn't set tabs properly; it can't intuit personal choices for interrupt and line kill characters, so it leaves these the old UNIX standards ^? (delete) for interrupt and @ for line kill.

It could well be argued that the shell should be responsible for insuring that the terminal remains in a sane state; this would eliminate the need for this program.

NOTES

For compatibility with earlier versions of *tset*, a number of flags are accepted whose use is discouraged:

−d type equivalent to **−m** dialup:type

−p type equivalent to **−m** plugboard:type

−a type equivalent to **−m** arpanet:type

−E c Sets the erase character to *c* only if the terminal can backspace.

− prints the terminal type on the standard output

−r prints the terminal type on the diagnostic output.

AUTHOR

Eric Allman

NAME

tsort − topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

EXAMPLE

```
ar cr library `lorder *.o | tsort`
```

intends to build a new library from existing .o files.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty − get the terminal's name

SYNOPSIS

tty [**−l**] [**−s**]

DESCRIPTION

Tty prints the path name of the user's terminal. The **−l** option prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line. The **−s** option inhibits printing of the terminal's path name, allowing one to test just the exit code.

EXAMPLE

```
tty
```

produces **/dev/tty7** if user is on **tty7**.

EXIT CODES

2 if invalid options were specified,
0 if standard input is a terminal,
1 otherwise.

DIAGNOSTICS

"not on an active synchronous line" if the standard input is not a synchronous terminal and **−l** is specified.
"not a tty" if the standard input is not a terminal and **−s** is not specified.

U3B(1) **SEE** ***MACHID*** **U3B(1)**

U3B5(1) **SEE** ***MACHID*** **U3B5(1)**

NAME

ul — do underlining

SYNOPSIS

ul [**−t** *terminal*] [*name* ...]

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining. If **−t** is present, *terminal* is used as the terminal kind. Otherwise, first the environment is searched, and if necessary, **/etc/termcap** is read to determine the appropriate sequences for underlining. If none of the fields **us**, **ue**, or **uc** are present, and if **so** and **se** are present, standout mode is used to indicate underlining. If the terminal can overstrike, or handles underlining automatically, *ul* behaves like *cat*(1). If the terminal cannot underline, underlining is ignored.

EXAMPLE

```
ul file1
```

displays "file1" on the terminal with underlined portions of the file either underlined, or in reverse video when this option is supported for the terminal.

FILES

/bin/cat	concatenate and print
/etc/termcap	terminal capability data base

SEE ALSO

man(1), nroff(1).

BUGS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

AUTHOR

Mark Horton

NAME

umask − set file-creation mode mask

SYNOPSIS

umask [000]

DESCRIPTION

The user file-creation mode mask is set to *000*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode 777 become mode 755; files created with mode **666** become mode **644**).

If *000* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

EXAMPLE

```
umask 22
```

sets file-creation mode mask such that at file creation, the *write* bits will be zeroed out for *group* and *other* users, regardless of mode specification in *create*.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

NAME

uname − print name of current UNIX system

SYNOPSIS

uname [**−snrvma**]

DESCRIPTION

Uname prints the current system name of the UNIX system on the standard output file. It is mainly useful to determine which system one is using. The options cause selected information returned by *uname*(2) to be printed:

−s print the system name (default).

−n print the nodename (the nodename may be a name that the system is known by to a communications network).

−r print the operating system release.

−v print the operating system version.

−m print the machine hardware name.

−a print all the above information.

EXAMPLE

```
    uname
```

on UniPlus$^+$ would print on the screen

```
    unix
```

SEE ALSO

uname(2).

NAME

unget − undo a previous get of an SCCS file

SYNOPSIS

unget [−**r**SID] [−**s**] [−**n**] files

DESCRIPTION

Unget undoes the effect of a **get** −**e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

−**r** *SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

−**s** Suppresses the printout, on the standard output, of the intended delta's *SID*.

−**n** Causes the retention of the gotten file which would normally be removed from the current directory.

EXAMPLE

```
% unget s.test1.c
1.2
```

undoes version 1.2 of "test1.c" set up for editing by an earlier **get -e**.

SEE ALSO

delta(1), get(1), help(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq − report repeated lines in a file

SYNOPSIS

uniq [**−udc** [+n] [−n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. 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.

EXAMPLE

```
uniq file1
```

prints contents of "file1" with adjacent identical lines removed.

SEE ALSO

comm(1), sort(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, as in the examples below.

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 (see the second example below).

Units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

EXAMPLE

```
You have:  inch
You want:  cm
           * 2.540000e+00
           / 3.937008e−01

You have:  15 lbs force/in2
You want:  atm
           * 1.020689e+00
           / 9.797299e−01
```

FILES

/usr/lib/unittab

SEE *PACK*

NAME

updater − update files between two machines

SYNOPSIS

updater [key] local remote ...

DESCRIPTION

Updater updates files between two machines.

One of the following key letters must be included:

t Take files from the remote machine, updating the local machine.

p Put files from the local machine onto the remote machine, updating the remote machine.

d List the difference between files on the local and remote machines.

The following key letters are optional:

u Update a file only if it exists on both machines; this is the default condition.

r Replace a file if it did not exist on the destination machine.

Local refers to the local directory name.

Remote refers to the remote directory names. Only one remote name can be specified if the **p** (put) key is specified.

ALGORITHM

Open **/dev/tty0** to the remote machine.

Stty the local port and send a stty command to the remote machine to condition both ends of the connection.

Send a "cd remote ; sumdir . | sort +2 > /tmp/rXXXXX" to remote machine for each remote system; "cd local ; sumdir . | sort > /tmp/lXXXXX" for local machine.

Wait for remote to complete.

Take **/tmp/rXXXXX.**

Do a comparison between the local and the union of the remotes:

```
exists on remote only:
      If both the t and r keys are specified, take the file; otherwise list
      the file.
exists on local only:
      If both p and r keys are specified, put the file; otherwise list the
      file.
exist on both but different:
      If t key is specified, take the file.
      If p key is specified, put the file.
      If d key is specified, list the file.
same:
      nothing
```

EXAMPLE

```
updater d . .
```

uses **/dev/tty0** to communicate with a remote machine and compares directories on the remote and local systems.

NOTES

This program is useful primarily to UniSoft.

NAME

uucp, uulog, uuname − UNIX system to UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file

uulog [options]

uuname [−l] [−v]

DESCRIPTION

Uucp.

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are willing to forward information.

The shell metacharacters ?, * and [...] appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by ~/*user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system, the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

If a simple ~*user* destination is inaccessible to *uucp*, data is copied to a spool directory and the user is notified by *mail*(1).

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*:

−d Make all necessary directories for the file copy (default).

−f Do not make intermediate directories for the file copy.

−c Use the source file when copying out rather than copying the file to the spool directory (default).

−C Copy the source file to the spool directory.

−m *file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

−n *user* Notify *user* on the remote system that a file was sent.

−e *sys* Send the *uucp* command to system *sys* to be executed there. (Note: this will only be successful if the remote machine allows the *uucp* command to be executed by **/usr/lib/uucp/uuxqt**.)

−r Queue job but do not start the file transfer process. By default a file transfer process is started each time uucp is evoked.

−j Control writing of the *uucp* job number to standard output (see below).

Uucp associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate the job.

The environment variable **JOBNO** and the **-j** option are used to control the listing of the *uucp* job number on standard output. If the environment variable **JOBNO** is undefined or set to **OFF**, the job number will not be listed (default). If *uucp* is then invoked with the **-j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time uucp is invoked. In this case, the **-j** option will suppress output of the job number.

Uulog

Uulog queries a summary log of *uucp* and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE**.

The options cause *uulog* to print logging information:

−s *sys* Print information about work involving system *sys*. If *sys* is not specified, then logging information for all systems will be printed.

−u *user* Print information about work done for the specified, *user*. If *user* is not specified, then logging information for all users will be printed.

Uuname.

Uuname lists the uucp names of known systems. The **−l** option returns the local system name. The **−v** option will print additional information about each system. A description will be printed for each system that has a line of information in **/usr/lib/uucp/ADMIN**. The format of ADMIN is:
sysname tab *description* tab.

EXAMPLE

```
uucp file1 unisoft! /usr/spool/uucppublic/file2
```

sends "file1" from the local machine, via the uucp network, to the "unisoft" machine, where it is saved as file "/usr/spool/uucppublic/file2".

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uux(1C), chgnod(1M), chmod(2).

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~**nuucp** or just ~).

NOTES

In order to send files that begin with a dot (e.g., .profile) the files must be qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

Uucp will not generate a job number for a strictly local transaction.

BUGS

All files received by *uucp* will be owned by *uucp*.
The **−m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **? • [...]** will not activate the **−m** option.
The **-m** option will not work if all transactions are local or if **uucp** is executed remotely via the **−e** option.
The **−n** option will function only when the source and destination are not on the same machine.
Only the first six characters of a *system-name* are significant. Any excess characters are ignored.

UULOG (1C) **SEE** ***UUCP*** **UULOG (1C)**

UUNAME (1C) **SEE** ***UUCP*** **UUNAME (1C)**

UUPICK (1C) **SEE** ***UUTO*** **UUPICK (1C)**

NAME

uustat − uucp status inquiry and job control

SYNOPSIS

uustat [options]

DESCRIPTION

Uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. The following *options* are recognized:

−j *jobn* Report the status of the *uucp* request *jobn*. If **all** is used for *jobn*, the status of all *uucp* requests is reported. An argument must be supplied otherwise the usage message will be printed and the request will fail.

−k *jobn* Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

−r *jobn* Rejuvenate *jobn*. That is, *jobn* is touched so that its modification time is set to the current time. This prevents *uuclean* from deleting the job until the jobs modification time reaches the limit imposed by *uuclean*.

−c *hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.

−u *user* Report the status of all *uucp* requests issued by *user*.

−s *sys* Report the status of all *uucp* requests which communicate with remote system *sys*.

−o *hour* Report the status of all *uucp* requests which are older than *hour* hours.

−y *hour* Report the status of all *uucp* requests which are younger than *hour* hours.

−m *mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.

−M *mch* This is the same as the *−m* option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.

−O Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each *uucp* request.

−q List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. Note that only one of the options **−j**, **−m**, **−k**, **−c**, **−r**, can be used with the rest of the other options.

For example, the command:

```
uustat −uhdc −smhtsa −y72
```

will print the status of all *uucp* requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
000001	the copy failed, but the reason cannot be determined
000002	permission to access local file is denied
000004	permission to access remote file is denied
000010	bad *uucp* command is generated
000020	remote system cannot create temporary file
000040	cannot copy to remote directory
000100	cannot copy to local directory
000200	local system cannot create temporary file
000400	cannot execute *uucp*
001000	copy (partially) succeeded
002000	copy finished, job deleted
004000	job is queued
010000	job killed (incomplete)
020000	job killed (complete)

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

FILES

/usr/spool/uucp	spool directory
/usr/lib/uucp/L_stat	system status file
/usr/lib/uucp/R_stat	request status file

SEE ALSO

uucp(1C).

NAME

uuto, uupick — public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [options] source-files destination
uupick [−ssystem]

DESCRIPTION

Uuto sends *source-files* to *destination*. *Uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *Logname* is the login name of someone on the specified system.

Two *options* are available:

−p	Copy the source file into the spool directory before transmission.
−m	Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/ *user*/ *mysystem*/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default).
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the −s*system* option will only search the PUBDIR for files sent from *system*.

EXAMPLE

```
uuto -p file1 file2 file3 ucbvax!Joe
```

would send the three files to user Joe on ucbvax

```
uupick [executed by Joe]
```

would tell him what has arrived and from where.

FILES

PUBDIR=/usr/spool/uucppublic public directory

NOTES

In order to send files that begin with a dot (e.g., .profile) the file must be qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C)
uuclean(1M) in the *UniPlus⁺ Administrator's Manual.*

NAME

uux – UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail* (1)) via *uux*.

The *command-string* is made up of one or more arguments that look like a Shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ˜*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

Any special shell characters such as **<>;|** should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

Uux will attempt to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses.

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine. Executable commands are listed in **/usr/lib/uucp/L.cmds** on the remote system. The format of the **L.cmds** file is:

cmd,machine1,machine2, . . .

If no machines are specified, then any machine can execute cmd. If machines are specified, only the listed machines can execute **cmd**. If the desired command is not listed in **L.cmds** then no machine can execute that command.

Redirection of standard input and output is usually restricted to files in PUBDIR. Directories into which redirection is allowed must be specified in **/usr/lib/uucp/USERFILE** by the system administrator. See the *UUCP* chapter of the *Administrator Guide*

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

– The standard input to *uux* is made the standard input to the *command-string*.

–n Send no notification to user.

–m*file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

–j Control writing of the *uucp* job number to standard output.

Uux associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate the job.

The environment variable **JOBNO** and the –j option are used to control the listing of the *uux* job number on standard output. If the environment variable **JOBNO** is undefined or set to **OFF**, the job number will not be listed (default). If *uucp* is then invoked with the –j option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time *uux* is invoked. In this case, the **–j** option will suppress output of the job number.

EXAMPLE

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"
```

will get the "f1" files from the usg and pwba machines, execute a *diff* command and put the results in "f1.diff" in the local directory.

```
uux a!uucp b!/usr/file \(c!/usr/file\)
```

will send a *uucp* command to system **a** to get /usr/file from system **b** and send it to system **c**.

FILES

/usr/lib/uucp/L.sys	List of system names and when to call them
/usr/lib/uucp/L-cmd	List of commands for *uuxqt* to execute
/usr/lib/uucp/L-devices	List of device codes and speeds
/usr/lib/uucp/L-dialcodes	List of phone numbers in L.sys
/usr/lib/uucp/SYSTEMNAME	Name of this system
/usr/lib/uucp/USERFILE	List of users and required pathname prefixes
/usr/lib/uucp/uucico	copy in, copy out program; called by *uucp*
/usr/lib/uucp/uuclean	spool directory cleanup program; called by *uucp*
/usr/lib/uucp/uuxqt	command execution program; called by *uucp*

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory (PUBDIR)

SEE ALSO

uucp(1C), mail(1), uuclean(1M).

BUGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

Only the first six characters of the *system-name* are significant. Any excess characters are ignored.

NAME

val − validate SCCS file

SYNOPSIS

val −
val [−s] [−rSID] [−mname] [−ytype] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a −, and named files.

Val has a special argument, −, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

−s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

−r*SID* The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e.g., **−r1** is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e.g., **−r**1.0 or **−r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

−m*name* The argument value *name* is compared with the SCCS %M% keyword in *file*.

−y*type* The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i.e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **−y** mismatch;
bit 7 = %M%, **−m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned − a logical **OR** of the codes generated for each command line and file processed.

EXAMPLE

```
val -
-yc -mabc s.abc
-mxyz -ypll s.xyz
```

first checks if file "s.abc" has a value *c* for its **type** flag and value *abc* for the **module** name flag. Once processing of the first file is completed, *val* then processes the remaining files (in this case "s.xyz") to determine if they meet the characteristics specified by the keyletter arguments associated with them.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc − version control

SYNOPSIS

vc [−a] [−t] [−cchar] [−s] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the **−t** keyletter (see below). The default control character is colon (:), except as modified by the **−c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The **−a** keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

−a Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.

−t All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.

−c*char* Specifies a control character to be used in place of :.

−s Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An asg statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

:if condition

⋮

:end Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

```
<cond>    ::= [ "not" ] <or>
<or>      ::= <and> | <and> "|" <or>
<and>     ::= <exp> | <exp> "&" <and>
<exp>     ::= "(" <or> ")" | <value> <op> <value>
<op>      ::= "=" | "!=" | "<" | ">"
<value>   ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```
= != > <   all of equal precedence
&
|
```

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

::text

Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the −a keyletter.

:on

:off

Turn on or off keyword replacement on all lines.

:ctl char
    Change the control character to char.

:msg message
    Prints the given message on the diagnostic output.

:err message
    Prints the given message followed by:
        ERROR: err statement on line ... (915)
    on the diagnostic output. *Vc* halts execution, and returns an exit code of 1.

EXAMPLE

If you have a file named "note" containing:

```
:dcl NAME,PLACE
        :NAME:,
        Just a note to remind you that we have a meeting
        scheduled Monday morning at :PLACE:.
```

the command

```
vc -a NAME=Joe PLACE=UniSoft < note
```

will produce

```
Joe,
Just a note to remind you that we have a meeting
scheduled Monday morning at UniSoft.
```

SEE ALSO

ed(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 − normal
1 − any error

SEE *VI*

NAME

version — reports version number of files

SYNOPSIS

version name ...

DESCRIPTION

Version takes a list of files and reports the version number. If the file is not a binary, it reports: "not a binary". If no version number is associated with the file, it reports: "No version header." *Version* also reports the object file format of the file, i.e., either "Coff object file format," or "Old a.out object file format." *Version* is useful for determining which version of the current program you are running.

EXAMPLE

```
version /bin/version
```

prints the version number of the version program.

NAME

vi − screen-oriented (visual) display editor based on *ex*

SYNOPSIS

vi [**−t** *tag*] [**−r** *file*] [**−l**] [**−w***n*] [**−x**] [**−R**] [**+***command*] name ...
view [**−t** *tag*] [**−r** *file*] [**−l**] [**−w***n*] [**−x**] [**−R**] [**+***command*] name ...
vedit [**−t** *tag*] [**−r** *file*] [**−l**] [**−w***n*] [**−x**] [**−R**] [**+***command*] name ...

DESCRIPTION

Vi (visual) is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file. The *Vi Quick Reference* card, and the **VI** section of the *User Guide* provide full details on using *vi*.

INVOCATION

The following invocation options are interpreted by *vi*:

−t *tag* Edit the file containing the *tag* and position the editor at its definition.

−r*file* Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.

−l LISP mode; indents appropriately for lisp code, the () {} [[and]] commands in *vi* and *open* are modified to have meaning for *lisp*.

−w*n* Set the default window size to *n*. This is useful when using the editor over a slow speed line.

−x Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

−R Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.

+*command* The specified *ex* command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

VI MODES

Command Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.

Input Entered by **a i A I o O c C s S R**. Text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.

Last line	Reading input for : / ? or !; terminate with CR to execute, interrupt to cancel.

COMMAND SUMMARY

Sample Commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
I*text***ESC**	insert *text*
cw*new***ESC**	change word to *new*
ea*s***ESC**	pluralize word
x	delete a character
dw	delete a word
dd	delete a line
3dd	... 3 lines
u	undo previous change
ZZ	exit *vi*, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*ex cmd***CR**	any *ex* or *ed* command

Counts Before Vi Commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**
scroll amount	**^D ^U**
repeat effect	most of the rest

Interrupting, Canceling

ESC	end insert or incomplete cmd
^?	(delete or rubout) interrupts
^L	reprints screen
^R	reprint screen if ^L is → key

File Manipulation

:wCR	write back changes
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:!*cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	to tag file entry *tag*
^]	**:ta**, following word is *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a **CR**.

Positioning Within File

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/*pat*	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/*pat*/+*n*	nth line after *pat*
?*pat*?−*n*	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }

Adjusting the Screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z−CR	... at bottom
z.CR	... at center
/*pat*/**z−CR**	*pat* line at bottom
z*n***.CR**	use *n* line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and Returning

``	move cursor to previous context
''	... at first non-white in line
m*x*	mark current position with letter *x*
`*x*	move cursor to mark *x*
'*x*	... at first non-white in line

Line Positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
−	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character Positioning

^	first non-white
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	f backward
t*x*	up to *x* forward
T*x*	back up to *x*
;	repeat last **f F t** or **T**
,	inverse of ;
|	to specified column
%	find matching **({)** or **}**

Words, Sentences, Paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back **W**
E	to end of **W**

Commands for LISP Mode

)	Forward s-expression
}	... but do not stop at atoms
(	Back s-expression
{	... but do not stop at atoms

Corrections During Insert

^H	erase last character
^W	erase last word
erase	your erase, same as **^H**
kill	your kill, erase input this line
****	quotes **^H**, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and Replace

a	append after cursor
i	insert before cursor
A	append at end of line

I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single character with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command
=	indent for LISP

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	... before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"*x***p**	put from buffer *x*
"*x***y**	yank to buffer *x*
"*x***d**	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d***p**	retrieve *d*th last delete

AUTHOR

Vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

ex (1).
Vi Quick Reference Card.
VI in the *User Guide.*

CAVEATS AND BUGS

Software tabs using ^T work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

SEE *VI*

NAME

wc − word count

SYNOPSIS

wc [**−lwc**] [names]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **−lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

EXAMPLE

```
wc filea fileb filec
```

reports the number of lines, words, and characters in each of the files.

NAME

what — identify SCCS files

SYNOPSIS

what [**−s**] files

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first " , > , new-line, \ , or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the SCCS command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

−s Quit after finding the first occurrence of pattern in each file.

EXAMPLE

If "test1.c" has the following string

```
char v[] = "@(#)1 test1.c 2";
```

typing

```
what test1.c
```

would print the following:

```
test1.c:
        1 test1.c 2
```

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use *help*(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

whereis – locate source, binary, and/or manual for program

SYNOPSIS

whereis [–sbm] [–u] [–SBMdir ...–f] name ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and standard extensions for source files and manual entries, e.g., ".c" and ".lm". Prefixes of "s." resulting from use of source code control are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places. If any of the **–b**, **–s**, or **–m** flags are given then *whereis* searches only for binaries, sources, or manual sections respectively (or any two thereof). The **–u** flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis –m –u *" asks for those files in the current directory which have no documentation.

Finally, the **–B**, **–M**, and **–S** flags may be used to change or otherwise limit the places where *whereis* searches. The **–f** file flags is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
whereis –u –M /usr/man/man1 –S /usr/src/cmd –f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

AUTHOR

William Joy

BUGS

Since the program uses *chdir*(2) to run faster, pathnames given with the --M, –S, and –B must be full; i.e., they must begin with a "/".

NAME

who − who is on the system

SYNOPSIS

who [−uTHlpdbrtasq] [file]

who am i

who am I

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the **/etc/utmp** file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

Who with the **am i** or **am I** option identifies the invoking user.

Except for the default **−s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

−u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (**.**) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

−T This option is the same as the **−u** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A **+** appears if the terminal is writable by anyone; a **−** appears if it is not. **Root** can write to all lines having a **+** or a **−** in the *state* field. If a bad line is encountered, a **?** is printed.

−l This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

−H This option will print column headings above the regular output.

−q This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

−p This option lists any non-getty process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

−d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(2)), of the dead process. This can be useful in determining why a process terminated.

−b This option indicates the time and date of the last reboot.

−r This option indicates the current *run-level* of the *init* process.

−t This option indicates the last change to the system clock (via the *date*(1) command) by **root**. See *su*(1).

−a This option processes **/etc/utmp** or the named *file* with all options turned on.

−s This option is the default and lists only the *name*, *line*, *time*, and *remote host* (if any) fields.

EXAMPLE

```
who am i
```

reports the name under which you are currently logged in.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), login(1), mesg(1), su(1), wait(2), inittab(4), utmp(4)
init(1M) in the *UniPlus⁺ Administrator's Manual*.

NAME

whoami — print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES

/etc/passwd	Name data base

SEE ALSO

who (1)

NAME

write − write to another user

SYNOPSIS

write user [line]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* **(tty??) [** *date* **]...**

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg*(1) command. Writing to others is normally allowed by default. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character ! is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

EXAMPLE

```
write unisoft tty7
```

writes unisoft on terminal 7, unless messages have been refused with *mesg*(1).

FILES

/etc/utmp	to find user
/bin/sh	to execute !

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), talk(1N), who(1).

DIAGNOSTICS

user is not logged in if the person you are trying to *write* to is not logged in.

Permission denied if the person you are trying to *write* to denies that permission (with *mesg*).

Warning: cannot respond, set mesg -y
if your terminal is set to *mesg n* and the recipient cannot respond to you.

Can no longer write to user
if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

xargs − construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **−i** flag). Flags **−i**, **−l**, and **−n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **−l** vs. **−n**), the last flag has precedence. *Flag* values are:

−l*number* *Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **−x** is forced.

−i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **−x** is also forced. **{}** is assumed for *replstr* if not specified.

−n*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **−x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

−t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

−p Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**−t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command.*

−x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.

−s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **−s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

−e*eofstr* *Eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if **−e** is not coded. The value **−e** with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical EOF string is encountered.

Xargs will terminate if either it receives a return code of **−1** from, or if it cannot execute, *command.* When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **−1**.

EXAMPLE

```
ls $1 | xargs −i −t mv $1/{} $2/{}
```

will move all files from directory $1 to directory $2, and echo each move command just before doing it.

```
(logname; date; echo $0 $*) | xargs >>log
```

will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file "log".

```
ls | xargs −p −l ar r arch
ls | xargs −p −l | xargs ar r arch
```

causes the user to be asked which files in the current directory are to be archived and archives them into "arch" one at a time in the first instance, or as in the second instance, many at a time.

```
echo $* | xargs −n2 diff
```

will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments.

SEE ALSO

sh(1).

DIAGNOSTICS

Self-explanatory.

NAME

xstr − extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**−c**] [**−**] [file]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

```
xstr −c name
```

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c,* to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

```
xstr
```

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

```
xstr name
```

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '−' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc −E name.c | xstr −c −
cc −c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' doesn't touch *strings*

SEE ALSO

mkstr(1)

AUTHOR

William Joy

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc — yet another compiler-compiler

SYNOPSIS

yacc [−vdlt] 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 amoiguities.

The output file, **y.tab.c,** must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **−v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **−d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **−l** flag is given, the code produced in **y.tab.c** will *not* contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **−t** option is used, this debugging code will be compiled by default. Independent of whether the **−t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a pre-processor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

EXAMPLE

```
yacc file1.y
```

invokes *yacc* to process file "file1.y" in *yacc*-format.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.debug, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1), malloc(3X).
YACC in the *Programming Tools Guide*.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

Colophon

Composed at UniSoft Systems Inc.
on the UniPlus^{+} Operating System
Designed by the Documentation Department
Printed in Times Roman on Sequoia Matt